EXTENSIÓN UML PARA EL MODELADO DE MAPAS NAVEGACIONALES DE APLICACIONES WEB BASADO EN MDA

HUMBERTO JAVIER CORTÉS BENAVIDES

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Máster en Sistemas Inteligentes

Fecha: 8 de septiembre de 2011

Director:

Antonio Navarro Martín

Autorización de Difusión

HUMBERTO JAVIER CORTÉS BENAVIDES

8 de septiembre de 2011

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "EXTENSIÓN UML PARA EL MODELADO DE MAPAS NAVEGACIONALES DE APLICACIONES WEB BASADO EN MDA", realizado durante el curso académico 2010-2011 bajo la dirección de ANTONIO NAVARRO MARTÍN en el Departamento de INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen

La aparición de las arquitecturas multicapas y las arquitecturas orientadas a servicios en el proceso de desarrollo de software, ha promovido que la capa de presentación de las aplicaciones en general, y de las aplicaciones Web en particular, sea totalmente independiente del resto de las capas de una aplicación. Sin embargo, las aplicaciones Web complejas pueden tener miles de páginas enlazadas y construidas usando diferentes tecnologías. Por lo tanto, la caracterización de mapas navegacionales es una labor cada vez más compleja. En este trabajo se presenta NMMp, una extensión UML que: (i) proporciona una visión abstracta de la estructura navegacional de la capa de presentación de las aplicaciones Web, con independencia de detalles arquitectónicos y lenguajes de programación; (ii) puede ser automáticamente transformada, siguiendo los principios MDA en modelos UML-WAE, los cuales pueden ser fácilmente integrados con el diseño del resto de las capas de una aplicación Web; (iii) promueve el uso de patrones arquitectónicos y de diseño; y (iv) ha sido desarrollada con estándares OMG, lo cual facilita su uso en la industria por medio de herramientas CASE UML de propósito general.

Palabras clave

MDA, MDD, NMM, QVT, Perfiles UML, Metamodelado.

Summary

The advent of multitier and service-oriented architectures in the software development process encourages that the presentation tier of the general applications and web applications is totality detached from the rest of the tiers of the application. However, complex web applications can have thousands of linked web pages built using different technologies; therefore, the characterization of the navigations maps is becoming more complex task. This work presents NMMp, a UML extension that: (i) provides an abstract vision of the navigational structure of the presentation tier of web applications, with independence of architectural details and programming languages; (ii) can be automatically transformed, following the MDA principles in UML-WAE models, which can be easily integrated with the design of the rest of tiers of the web application; (iii) encourages the use of architectural and design patterns; and (iv) has been developed with OMG standards, which facilitates its use with general purpose UML CASE tools by in industry.

Keywords

MDA, MDD, NMM, QVT, UML Profile, Metamodeling.

Índice de contenidos

Indice de figuras

Índice de tablas

Agradecimientos

Es una enorme satisfacción tener la oportunidad de presentar este trabajo porque con él siento que he logrado cumplir parte de un reto profesional y personal consistente en reconciliar los conocimientos académicos, teóricos, abstractos con la aplicación industrial, práctica y concreta.

Por ello, quiero expresar mi más profundo agradecimiento a Antonio Navarro Martín, director de este proyecto, por toda su dedicación tanto en su dirección como ejecución, y por haberme compartido sin ninguna reserva parte de su conocimiento y experiencia.

Quiero agradecer y dedicar este trabajo a mi Madre por su constante e inquebrantable apoyo, a Tatiana y Ángela por su paciencia y cariño y a mis hermanos por su colaboración durante la ejecución de este proyecto.

A todos muchas gracias.

1. Introducción

En general, el diseño de interfaces de usuario es una labor compleja. Sin embargo, esta complejidad crece sustancialmente en el contexto de las aplicaciones Web donde las interfaces de usuario pueden contener fácilmente cientos de páginas conectadas a través de miles de enlaces. Así mismo, no es difícil encontrar aplicaciones Web compuestas por una mezcla de páginas estáticas y dinámicamente creadas, donde estas últimas sean construidas usando diferentes tecnologías como J2EE (Weaver et al., 2004), ASP (Spaanjaars, 2010), PHP (Thompson and Nowicki, 2009), etc.

La comunidad de la Ingeniería Web ha aportado el concepto de *mapas navegacionales* para tratar de gestionar la complejidad inherente a la capa de presentación de las aplicaciones Web. Los mapas navegacionales describen una aplicación Web desde un punto de vista global (Abrahao et al., 2003) exponiendo todas las posibles secuencias de páginas por las que puede navegar un usuario. Por este motivo su caracterización es un punto clave durante el proceso de desarrollo de una aplicación Web (Navarro et al., 2005). Por medio del uso de mapas navegacionales tanto desarrolladores como usuarios pueden beneficiarse. Los desarrolladores pueden obtener una perspectiva completa de una aplicación Web mientras que los usuarios pueden navegar y encontrar la información deseada eficientemente (Navarro et al., 2008a).

En el presente, la mayoría de las notaciones en Ingeniería Web proporcionan un tratamiento integral para el diseño y desarrollo de aplicaciones Web, acoplando la estructura de los mapas navegacionales con el desarrollo del resto de la aplicación. Este enfoque tiene la ventaja de caracterizar las aplicaciones Web como un “todo”, facilitando la generación de aplicaciones Web *listas* para ejecutarse (Bozzon et al., 2006; Distante et al., 2007; Koch and Kraus, 2003; Melia et al., 2010).

Sin embargo, este enfoque no tiene en cuenta el uso de patrones arquitectónicos y de diseño para el desarrollo de aplicaciones multicapas y aplicaciones SOA (*Service-Oriented Applications*) utilizados exitosamente en la industria (Alur et al., 2003; Erl, 2009a; Fowler, 2003; Monday, 2003). Estos patrones afectan el diseño de cada una de las capas proporcionado reglas de diseño para la definición e interconexión de cada una de ellas.

Desde este punto de vista, se puede decir que existen dos enfoques para el diseño de aplicaciones Web: (i) el uso de notaciones de diseño proporcionadas por la comunidad de la

Ingeniería Web que, mediante el uso de herramientas *CASE específicas* tienen la capacidad de generar una aplicación Web; y (ii) el uso de patrones arquitectónicos y de diseño que, mediante el uso de herramientas *CASE genéricas* tienen la capacidad de generar *parte* de una aplicación Web. Durante el desarrollo de este trabajo se ha llamado *enfoque basado en notación* a la primera solución y *enfoque basado en patrones* a la segunda.

Observando las fuentes de los dos enfoques se evidencia que el sector de la academia está más cerca del enfoque basado en notación (Bozzon et al., 2006; Distante et al., 2007; Koch and Kraus, 2003; Linaje et al., 2007; Melia et al., 2010), mientras que el sector de la industria se aproxima al enfoque basado en patrones (Alur et al., 2003; Erl, 2009b; Fowler, 2003; Monday, 2003).

Un punto a destacar sobre estos dos enfoques es su desarrollo ortogonal. La mayoría de los enfoques basados en notación no mencionan los conceptos de patrones arquitectónicos para aplicaciones multicapas ni para aplicaciones SOA. De igual forma, los patrones arquitectónicos no tienen en cuenta las notaciones de la Ingeniería Web para el diseño de las aplicaciones.

También se podría concebir los dos enfoques como dos niveles diferentes de abstracción: los enfoques basados en notación caracterizan diseños *abstractos* de aplicaciones Web, mientras que el enfoque basado en patrones caracterizan diseños *detallados* de aplicaciones Web. Sin embargo, el desarrollo de una aplicación Web implica tanto labores de diseño abstracto como de desarrollo concreto de código, de esta manera: (i) si se utiliza el enfoque basado en notación, el diseño es construido usando una notación de diseño concreta y no se garantiza la presencia de patrones arquitectónicos multicapas ni SOA implementados en el código; y (ii) si se utiliza el enfoque basado en patrones, la notación de diseño más razonable es UML (la mayor parte de los patrones arquitectónicos son descritos usando UML (Alur et al., 2003; Fowler, 2003; Monday, 2003) y estos patrones son implementados en el código de la aplicación.

Por tanto, aunque se pueden considerar los enfoques basados en notación y basados en patrones como dos niveles diferentes de abstracción, en la práctica, ellos representan dos filosofías diferentes que llevan a procesos de desarrollo diferentes.

El enfoque basado en patrones garantiza que las aplicaciones Web desarrolladas sean flexibles y mantenibles. El dividir una aplicación Web en capas separadas que desempeñan diferentes roles y funcionalidades permite cambiar un aspecto de la aplicación (por ejemplo, en presentación, lógica o integración) minimizando los efectos en el resto de aspectos. La

mantenibilidad de las arquitecturas multicapas ha sido probada durante años en la industria (Alur et al., 2003; Erl, 2009b; Fowler, 2003; Monday, 2003). Además admite diferentes tipos de despliegue permitiendo que diferentes capas sean situadas de forma flexible en diferentes servidores o clientes, y, sobre todo, nos proporciona una clara delimitación y situación de dónde debe estar cada tipo de componente funcional e incluso cada tipo de tecnología.

Otra ventaja del enfoque basado en patrones es que permite caracterizar el diseño de las aplicaciones Web a través de diagramas UML desarrollados con herramientas CASE genéricas. Esta característica facilita el diseño de las aplicaciones dada la naturaleza estándar de la notación UML, además, facilita la evolución de las aplicaciones dado que el mantenimiento del código no esta sujeto a ninguna herramienta específica.

Sin embargo, en nuestra opinión, el enfoque basado en patrones puede beneficiarse también de las ventajas de los enfoques basados en notaciones. En particular, la capa de presentación puede ser un buen punto de partida para aplicar notaciones de diseño específicas. La capa de presentación de las aplicaciones Web puede contener miles de elementos (páginas Web y enlaces) que deberían ser representados de la forma más abstracta posible para facilitar el entendimiento tanto de desarrolladores como de usuarios.

Los usuarios no pueden validar la arquitectura software de una aplicación Web, pero si podrían validar la estructura navegacional de la aplicación. Los desarrolladores necesitan conocer la interconexión de las páginas Web para lograr implementarlas. Por lo tanto, los *mapas navegacionales* parecen ser una herramienta muy valiosa para todos los implicados en el desarrollo de una aplicación Web.

Este trabajo ha tomado como base la notación de diseño NMM (*Navigation Maps Modeling*) (Navarro et al., 2008a) desarrollada para caracterizar mapas navegacionales de aplicaciones Web. NMM tiene un desarrollo formal y su propia herramienta CASE, un modelo NMM puede ser fácilmente transformado en un modelo UML-WAE (*Web Application Extension*) (Conallen, 1999). Sin embargo, la falta de integración de la notación NMM con herramientas CASE genéricas ha sido un serio obstáculo en este proceso de transformación.

Para salvar este obstáculo, se ha desarrollado NMMp (*NMM profile*), una extensión UML basada en NMM que se ha implementado utilizando el concepto de *Perfiles UML* (OMG-UML, 2010a), con el objetivo de ser utilizado en un enfoque mixto entre el enfoque basado en notación y el enfoque basado en patrones. Desde el punto de vista del enfoque basado en notación, NMMp

proporciona una visión abstracta de la estructura navegacional de la capa de presentación de las aplicaciones Web con independencia de detalles arquitectónicos y de programación. Desde el punto de vista del enfoque basado en patrones, los diagramas NMMp pueden ser combinados con diagramas UML para modelar el resto de capas permitiendo incluir patrones arquitectónicos.

Asi como en el caso de NMM, los modelos NMMp pueden ser transformados fácilmente en modelos UML-WAE, los cuales pueden a su vez ser directamente integrados con el diseño del resto de las capas de una aplicación Web. Para implementar esta transformación se ha decidido aplicar el enfoque MDA (*Model-Driven Architecture*) (OMG-MDA, 2003) que, para este caso, consiste en definir transformaciones automáticas entre un modelo NMMp y un modelo UML-WAE. De esta forma, a partir del modelo NMMp de un sistema, acorde a un perfil UML y contando con el perfil y la descripción del modelo destino UML-WAE y con una serie de reglas de transformación, se pueda obtener el modelo UML-WAE con la arquitectura seleccionada de la forma más automatizada posible.

Así, de acuerdo al enfoque MDA, los modelos NMMp son *Modelos Independientes de la Plataforma (PIMs)* (OMG-MDA, 2003) que omiten detalles arquitectónicos, mientras los modelos UML-WAE generados son PIMs que modelan arquitecturas concretas y que además pueden ser fácilmente transformados a *Modelos Específicos de la Plataforma (PSMs)* (OMG-MDA, 2003).

La característica de poder transformar automáticamente modelos NMMp a modelos UML-WAE reconcilia el enfoque basado en notaciones con el enfoque basado en patrones, ya que: (i) los modelos UML-WAE generados usan en si mismos patrones arquitectónicos multicapas y SOA; y (ii) los patrones de diseño adicionales de presentación, de negocio y de integración pueden ser usados en el diseño del resto de las capas.

De esta forma, NMMp puede ser utilizado como complemento para modelar aplicaciones Web con UML, el cual se ha convertido en un estándar reconocido tanto en el sector académico como industrial para modelar lógica de negocio, capas de integración y capas de recursos.

Finalmente, NMMp se ha desarrollado utilizando estándares OMG (OMG, 2010), lo cual facilita su uso industrial con herramientas CASE tanto comerciales como abiertas. En particular se ha utilizado la herramienta CASE *Borland Together* (Borland, 2011) por su soporte integrado para *XMI* (OMG-XMI, 2007) y *QVT operacional mappings* (OMG-QVT, 2009) que permite definir las reglas de transformación entre modelos. El soporte a XMI y QVT permite la fácil

transición desde Borland Together a cualquier otra herramienta CASE que soporte estos estándares.

Las aportaciones de este trabajo son consecuencia directa de la evolución con respecto a NMM. Así, NMMp: (i) introduce algunas modificaciones al modelo subyacente NMM para hacerla más usable; (ii) hace evolucionar NMM de una notación formal a un extensión UML, incrementando su aplicabilidad; y (iii) define las aproximaciones basadas en notaciones en modelos para el desarrollo de aplicaciones web, convirtiendo a NMMp en uno de los mejores representantes de la aproximación mixta. Además, NMMp: (i) proporciona una visión abstracta de la estructura navegacional de la capa de presentación de las aplicaciones Web, con independencia de detalles arquitectónicos y lenguajes de programación; (ii) puede ser automáticamente transformada, siguiendo los principios MDA en modelos UML-WAE, los cuales pueden ser fácilmente integrados con el diseño del resto de las capas de una aplicación Web; (iii) promueve el uso de patrones arquitectónicos y de diseño; y (iv) ha sido desarrollada con estándares OMG, lo cual facilita su uso en la industria por medio de herramientas CASE UML de propósito general.

El trabajo realizado en este proyecto es parte del proyecto *Arquitecturas Avanzadas en Campus Virtuales*, AACV (TIN TIN2009-14317-C03-01). Dicho proyecto busca definir una arquitectura software que soporte las necesidades de los campus virtuales de nueva generación (Navarro et al., 2010). La arquitectura propuesta se basa en: (i) el uso de una arquitectura orientada a servicios (Erl, 2009a) para independizar los campus virtuales de las plataformas de *e-learning* subyacentes; y (ii) el uso de MDA para facilitar los cambios en los campus virtuales acorde a las demandas de los usuarios. Precisamente, NMMp tiene el objetivo de facilitar tanto el diseño de aplicaciones web en general, y campus virtuales en particular, favoreciendo la evolución de los mismos.

Este trabajo se divide en distintos Capítulos. El Capitulo 2 revisa el estado del arte, contextualizando el trabajo presentado en esta memoria. El Capitulo 3 presenta una discusión sobre la elección de perfiles o metamodelos para definir una notación visual. El Capitulo 4 define las transformaciones QVT de NMMp a diagramas UML WAE Model 1 (Brown et al., 2002). El Capitulo 5 define las transformaciones QVT de NMMp a diagramas UML WAE Model 2 (Brown et al., 2002). Finalmente se presentan las conclusiones y trabajo futuro. También se incluyen dos apéndices con la totalidad de las transformaciones QVT.

2. Estado del Arte

En esta sección se analizan las principales aproximaciones para el modelado de aplicaciones web y las nociones fundamentales de la arquitectura multicapa. Ambos elementos son necesarios para contextualizar las aportaciones de este trabajo, que proporciona una notación para el modelado de aplicaciones web en el seno de una arquitectura multicapa.

2.1 Aproximaciones para el Modelado de Aplicaciones Web

En esta sección se analizan diversas aproximaciones para el modelado de aplicaciones Web. Cabe destacar que estas aproximaciones podrían agruparse en dos categorías bien diferenciadas: unas se centran en proporcionar notaciones de modelado con soporte CASE, mientras que otras son herramientas CASE y/o entornos de desarrollo visuales centradas en la construcción de aplicaciones Web utilizando una notación visual. Podríamos decir que las primeras parten de *Modelos Independientes de la Plataforma* (OMG-MDA, 2003) para generar código, mientras que las segundas generan código a partir de *Modelos Específicos de la Plataforma* (OMG-MDA, 2003). A la primera categoría pertencen OOH, UWE y WebML (Gomez et al., 2001; Hennicker and Koch, 2001; Ceri et al., 2000), mientras que a la segunda categoría pertenecen IBM RSA y Oracle ADF (IBM, 2010; Oracle, 2010d). UML WAE (Conallen, 2003) y NMM (Navarro et al., 2008b) podrían estar a medio camino entre ambas. UML WAE permite representar modelos independientes de la plataforma, pero dependientes de la arquitectura para modelar aplicaciones Web. NMM permite representar modelos independientes de la plataforma, pero incluye reglas de transformación para generar modelos UML WAE.

A pesar de representar enfoques distintos para el modelado y desarrollo de aplicaciones Web, se analizan estas aproximaciones, ya que son de las más utilizadas por la academia y la industria para el modelado y construcción de este tipo de aplicaciones.

2.1.1 Object-Oriented Hypermedia – OOH

Object-Oriented Hypermedia – OOH (Gomez et al., 2001) es un método de modelado genérico que proporciona la semántica y notación necesaria para el desarrollo de aplicaciones Web. Extiende el conjunto de elementos gráficos de Object-Oriented Method OOM (Pastor et al., 1997), una metodología de desarrollo de software basada en el paradigma de la programación

automática y en los principios de modelado orientado a objetos, su principal característica es que el proceso de desarrollo hace énfasis en la etapa del modelado conceptual conforme con la metodología UML (OMG-UML, 20110).

OOH permite a los diseñadores concentrarse en los conceptos necesarios para definir interfaces Web compatibles con los modelos generados previamente con el método OOM. Para esto, el método OOH define nuevos diagramas a nivel conceptual que especifican las características semánticas y de notación para cubrir los aspectos navegacionales y de presentación de las aplicaciones Web. Además, OOH hace posible la generación de la arquitectura Web de forma automática, el código generado puede ser visto como una interfaz Web que se conecta a los módulos lógicos de una aplicación desarrollada a través del método OOM.

Los diagramas complementarios que OOH propone son: *Diagrama de Acceso Navegacional (NAD)*, que define la perspectiva navegacional, y el *Diagrama de Presentación Abstracta (APD)* que define los conceptos relacionados a la presentación Web. Estos dos tipos de diagrama capturan la información de diseño relacionada con la interfaz Web con la ayuda de un conjunto de patrones definidos en el *Catálogo de Patrones de Interfaz* propuestos también por el método OOH.

Proceso de desarrollo OOH

El proceso de desarrollo OOH define las etapas que el diseñador tiene que cubrir para construir una interfaz Web que cumpla todos los requisitos. El proceso comienza definiendo los Diagramas de Acceso Navegacional a partir del diagrama de clases UML modelado a través del método OOM. Ya que cada tipo de usuario tiene una vista diferente del sistema, es necesario definir para cada tipo de usuario al menos un Diagrama de Acceso Navegacional diferente.

Una vez los Diagramas de Acceso Navegacional son construidos, se puede generar a partir de estos una interfaz Web por defecto de forma automática. Esta interfaz por defecto, por su bajo nivel de sofisticación tanto desde el punto de vista visual como de usabilidad suele considerarse un prototipo. Para mejorar la calidad de esta interfaz, OOH introduce el segundo tipo de diagrama, el Diagrama de Presentación Abstracta, el cual está basado en el concepto de *Plantillas* (Fraternali and Paolini, 1998) y cuya estructura por defecto también puede ser directamente derivada desde los Diagramas de Acceso Navegacional. Como ayuda para refinar la

estructura navegacional y de presentación, mejorando la calidad, OOH propone el catálogo de patrones de interfaz que contiene un conjunto de constructores que han probado ser soluciones efectivas a problemas identificados en ambientes Web.

Una vez el Diagrama de Presentación Abstracta ha sido refinado, la interfaz de la aplicación Web puede ser generada de forma automática para el ambiente deseado (HTML, WML, ASP, JSP, etc.). Esta independencia de la tecnología de implementación final es una característica destacable del proceso de desarrollo OOH dada la continua evolución de lenguajes y tecnologías en el ambiente Web.

Diagramas de acceso navegacional

El modelo navegacional en OOH es representado a través de uno ó más diagramas de acceso navegacionales. Es necesario construir tantos diagramas de acceso navegacionales como vistas del sistema sean requeridas, por regla general, se necesita al menos un diagrama de acceso navegacional por cada tipo de usuario que navegará a través del sistema.

La construcción de un diagrama de acceso navegacional implica filtrar y enriquecer la información proporcionada por el diagrama de clases previamente modelado a través de OOM durante la fase de modelado conceptual. Para filtrar y enriquecer el diagrama de clases se debe tener en cuenta aspectos como el conjunto de objetos que pueden ser visitados a través de la navegación, el orden, cardinalidad y la relación entre el tipo de usuario y estos objetos. Un diagrama de acceso navegacional esta basado en cuatro tipos de constructores: *clases navegacionales*, *destinos navegacionales*, *enlaces navegacionales* y *colecciones*. A continuación detallamos brevemente estos conceptos:

- Clases navegacionales: están basadas en las clases del modelo conceptual, pueden ser vistas como clases del dominio cuya visibilidad de sus atributos y métodos han sido enriquecidos de acuerdo a los permisos de acceso y a los requisitos navegacionales. Un ejemplo de enriquecimiento es la declaración de tres tipos de atributos: Atributos V (atributos siempre visibles), Atributos R (atributos referenciados a los cuales se tendrá acceso bajo demanda del usuario) y Atributos H (atributos ocultos).
- Destinos navegacionales: un destino navegacional es un conjunto de clases navegacionales las cuales proporcionan, juntas, un requisito funcional para un tipo de usuario. Como podemos ver, OOH basa el modelo navegacional en los requisitos de

usuario sin tener en cuenta si la información es presentada en una simple página Web ó dividida en varias.

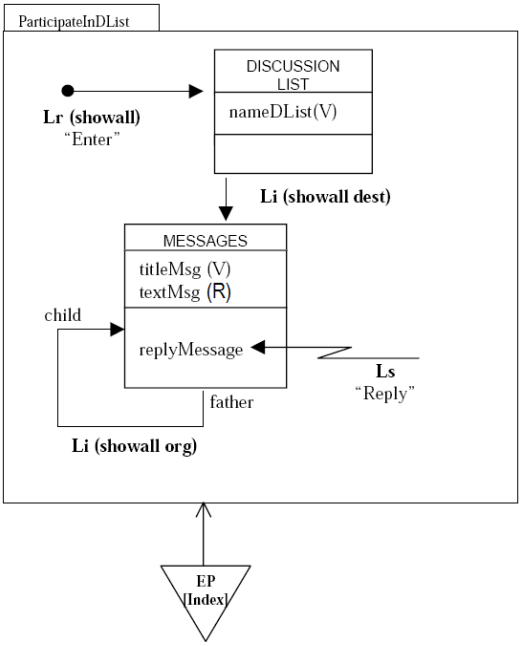
- Enlaces navegacionales: definen los caminos navegacionales que un tipo de usuario puede seguir a través del sistema. OOH define cuatro tipos de enlaces: Enlaces I (enlaces internos que definen el camino navegacional dentro de los límites de un destino navegacional), Enlaces T (enlaces transversales que son definidos entre clases navegacionales que pertenecen a diferentes destinos navegacionales), Enlaces R (enlaces de requisito que indicar el punto de entrada de navegación en un destino navegacional), Enlaces S (enlaces de servicio que muestran los servicios disponibles para un tipo de usuario).
- Colecciones: se representan por medio de un triángulo invertido, es una estructura que puede ser jerárquica que abstrae algunos conceptos relacionados con navegación interna o externa, modelan como la información de la interfaz de usuario es agrupada para hacer esta más accesible al usuario final.

La figura 2-1 presenta un ejemplo de un diagrama de acceso navegacional que modela un requisito navegacional de un sistema de administración de listas de discusión. Cada ítem de la lista de discusión está formado por un conjunto de mensajes ordenados jerárquicamente. Los usuarios del sistema pueden leer y replicar cualquier mensaje.

El requisito navegacional mostrado en la figura 2-1 es modelado a través del destino navegacional llamado “ParticipateInDList”, el cual contiene la información y los servicios necesarios para que los usuarios puedan navegar efectivamente a través de los ítems de las listas de discusión y los mensajes asociados.

El destino navegacional “ParticipateInDList” está compuesto por las vistas de dos clases del dominio: “DiscussionList” y “Messages”. La primera tiene un solo atributo llamado “nameDList” al que se le ha aplicado el modificador “V” (atributo siempre visible), la razón es que este atributo identifica unívocamente cada ítem de la lista de discusión y por lo tanto debe estar siempre presente. La clase “Messages” contiene le atributo “titleMsg” con el modificador “V” y el atributo “textMsg” con el modificador “R” (atributo referenciado), lo cual implica que el usuario tiene que explícitamente hacer clic en esta referencia para acceder al texto del mensaje.

Figura 2.1 Ejemplo de un diagrama de acceso navegacional



También podemos observar en la figura 2-1 que las clases "DiscussionList" y "Messages" se relaciona a través de un enlace I (enlace interno), este enlace tiene asociado el patrón navegacional "showall", lo cual causa que todas las instancias de la clase "Messages" contenidas en una instancia de la clase "DiscussionList" aparezcan juntas. El modificador "dest" que se aplica al patrón navegacional "showall" indica el hecho de que navegar a través del enlace implica un paso más en el proceso navegacional, es decir que el nombre de la lista de discusión y los mensajes relacionados a ella aparecerán en diferentes páginas. En el enlace R (enlace de requisito) etiquetado con "Enter" apunta a la clase navegacional que iniciará el proceso que cumple con el requisito del destino navegacional. Finalmente, el enlace S (enlace de servicio) es asociado con el servicio "replyMessage", lo cual modela la funcionalidad que permite a los usuarios responder los mensajes.

Diagramas de presentación abstracta

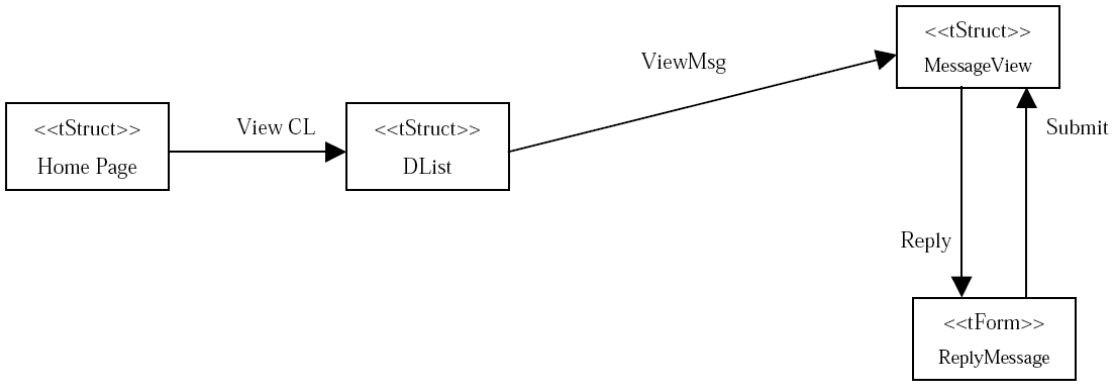
Como se comento anteriormente, OOH adopta el concepto de plantilla para especificar la apariencia visual así como la estructura de las páginas Web. Para separar los diferentes aspectos que constituyen la apariencia y el comportamiento final de la interfaz Web, se han definido cinco tipos de plantillas expresadas en XML (W3C, 2011a):

- Plantilla de estructura (*tStruct*): plantilla para definir la información que aparecerá en la página abstracta.
- Plantilla de estilo (*tStyle*): plantilla para definir las características tales como ubicación física de los elementos, tipografía o paleta de colores.
- Plantilla de formulario (*tForm*): plantilla para definir que elementos de datos son requeridos desde el usuario para interactuar con el sistema.
- Plantilla de funcionalidad (*tFunction*): plantilla para definir la funcionalidad del lado del cliente. La funcionalidad definida está basada en la especificación de *Document Object Model* (DOM) (W3C, 2005).
- Plantilla de ventana (*tWindow*): plantilla para definir conjuntos de vistas simultaneas disponibles para un usuario.

Los diagramas de presentación abstracta pueden ser derivados de la información capturada en los diagramas de acceso navegacional combinada con los valores por defecto definidos en las plantillas. En (Gomez et al., 2001) se muestra los pasos definidos en OOH para mapear los diferentes elementos contenidos en los diagramas de acceso navegacional a los elementos correspondientes en los diagramas de presentación abstracta. Este proceso da como resultado un diagrama de presentación abstracta por defecto, del cual se puede obtener una simple interfaz Web funcional. Normalmente esta simple interfaz Web requiere refinamiento para ser incluida en una aplicación final, su verdadero valor está en servir como prototipo en el cual validar que los requisitos de usuario estén correctamente capturados.

El proceso de refinamiento de un diagrama de presentación abstracta por defecto consiste en la modificación de su estructura, la forma más fácil de llevarlo a cabo es mediante la aplicación de los patrones definidos en el catálogo de patrones de OOH. En la figura 2-2 se muestra el diagrama de presentación abstracta por defecto derivado desde el diagrama de acceso navegacional presentado en la figura 2-1.

Figura 2-2. Ejemplo de un diagrama de presentación abstracta



Modelado de procesos de negocio en OOH

En (Koch and Kraus, 2003) se presenta el soporte que proporciona OOH para el modelado de procesos de negocio, que al igual que en muchos otros enfoques, en OOH está dirigido por los requisitos de usuario.

OOH basa el modelo de procesos de negocio en el concepto de servicio, visto como un conjunto de interfaces cuyo propósito es agrupar y caracterizar grupos de operaciones, teniendo en cuenta que la interfaz de usuario es responsable no solo de invocar cada operación del servicio, sino además guiar al usuario a través de un flujo de control predefinido.

En OOH el modelado de procesos de negocio consiste en cuatro pasos:

- Refinar el modelo conceptual enriqueciendo el diagrama de clases UML con una lista exhaustiva de atributos y métodos para satisfacer los procesos de negocio.
- Construir diagramas de actividades teniendo en cuenta las operaciones subyacentes. En la figura 2-3 se muestra un ejemplo de un diagrama de actividad para un proceso de negocio.
- Refinar el diagrama de acceso navegacional aplicando un conjunto de reglas predefinidas para que refleje la información de los procesos de negocio contenida en los diagramas de actividad. La figura 2-4 muestra una tabla con un conjunto de reglas de mapeo entre los elementos de un diagrama de actividad y los elementos de un diagrama de acceso navegacional.

- Refinar el diagrama de presentación. Como hemos visto, en OOH la información relacionada con la navegación entre procesos es modelada en los diagramas navegacionales, en los diagramas de presentación solo se presentan características visuales de los caminos navegacionales, por ejemplo, especificación explicita del camino de los enlaces pertenecientes al mismo proceso.

Figura 2-3. Diagrama de actividad para el proceso de negocio "Checkout"

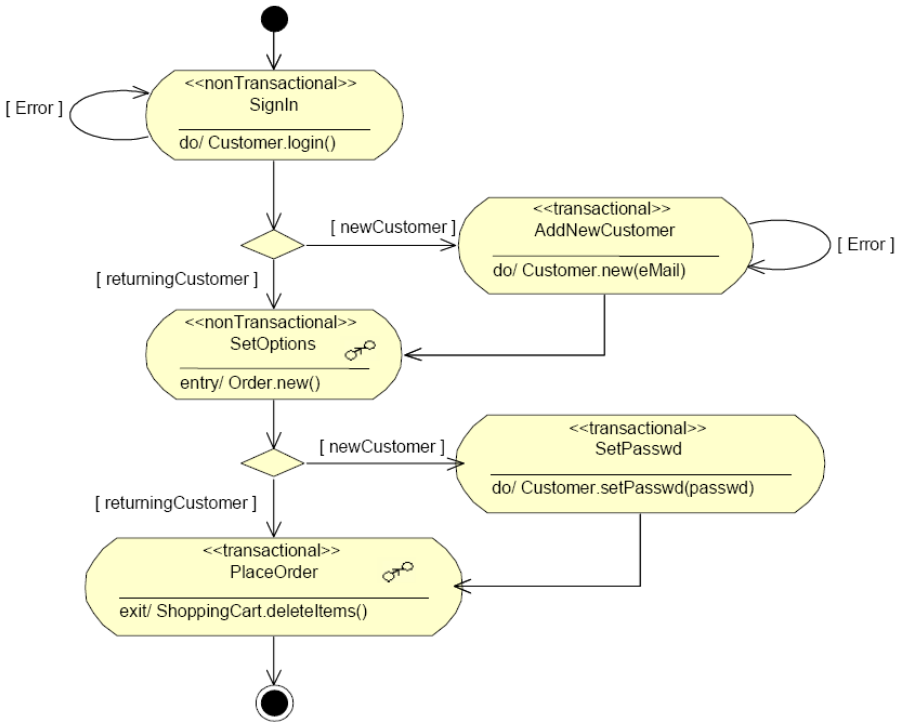


Figura 2-4. Reglas de mapeo entre las vistas de proceso y navegacional

Activity Diagram Element	NAD diagram element
Non-Transactional Activity	Navigational link refined with precondition filter
Transactional Activity	Service link associated with a Navigational class
Transition	Traversal link
Subactivity	Navigation Target
Branch	Collection from which a set of Traversal links with exclusive filters departs
Merge	Collection at which a set of Traversal links with no filters arrives.
Split-Join	Default path that traverses the concurrent activities sequentially from left to right

Modelado de aplicaciones RIA con OOH

En (Melia et al., 2009) se presenta OOH4RIA como un proceso de desarrollo de software dirigido por modelos que pretende cubrir todas las fases de desarrollo para la especificación de aplicaciones RIA (*Rich Internet Application*). Se basa en una extensión y mejora de la metodología OOH, a la que se ha agregado los modelos y transformaciones necesarias para obtener aplicaciones RIA.

Partiendo del modelo conceptual y navegacional de OOH se definen transformaciones modelo a texto para obtener la implementación del lado servidor de la aplicación, y transformaciones modelo a modelo para obtener los esquemas del modelo de presentación. Los modelos de presentación así obtenidos pueden ser vistos como representaciones estructurales de los componentes visuales que constituyen la interfaz de usuario RIA.

Los modelos de presentación están compuestos por un conjunto de pantallas denominadas *Screenshots* que permiten representar la organización de los diferentes *widgets* en un estado de la interfaz. Es importante destacar que estos *widgets* tienen la capacidad de enviar y obtener información del lado servidor de la aplicación, por lo tanto es necesario establecer una relación entre el *widget* y un elemento del modelo de navegación.

Debido a que las aplicaciones RIA poseen interfaces de usuario con interactividad similar a las interfaces de usuario de aplicaciones de escritorio, es necesario completar las características estáticas con un modelo que permita especificar la interacción entre los *widgets* y el resto del sistema. Este modelo es llamado *Modelo de Orquestación* y se representa como una máquina de estados, donde se define un estado para cada *screenshot* que internamente tiene sub-estados que representan los *widgets* con el comportamiento dinámico comentado. Este modelo es explicado en profundidad en (Perez et al., 2008). El último paso consiste en obtener la implementación del lado cliente de la aplicación que se logra mediante la aplicación de transformaciones modelo a texto.

Para facilitar el proceso anteriormente descrito se ha desarrollado la herramienta *OOH4RIA Tool* como un plugin del entorno Eclipse (Eclipse, 2011b), que se fundamenta en el framework *Graphical Modeling Framework* (GMF) (Eclipse, 2011a) para definir la parte del modelado y en el framework OpenArchitectureWare (OpenArchitectureWare, 2011) para la implementación de las transformaciones en el proceso de desarrollo dirigido por modelos definido por OOH4RIA.

Las aplicaciones RIA obtenidas mediante el proceso de desarrollo OOH4RIA están basadas en dos importantes frameworks Java. La interfaz de usuario RIA obtenida es implementada mediante *Google Web Toolkit* (GWT) (Google, 2010), que es un framework AJAX (Hudson, 2007) desarrollado por Google que permite crear la interfaz de usuario RIA escribiendo el código de navegador (HTML y JavaScript) directamente en Java. Por otro lado, la implementación del lado servidor de la aplicación se ha centrado en Hibernate (Elssamadisy, 2006), que permite un mapeo objeto-relacional permitiendo definir las reglas de negocio en un modelo orientado a objetos y hacerlos persistentes en una base de datos relacional.

2.1.2 UML-Based Web Engineering – UWE

UML-based Web Engineering – UWE (Hennicker and Koch, 2001) es un proceso de desarrollo para aplicaciones Web. UWE promueve el diseño sistemático y la generación semiautomática de aplicaciones usando exclusivamente técnicas, notación y mecanismos de extensión UML (OMG-UML, 2010a). Además, UWE cubre completamente el ciclo de vida del desarrollo de las aplicaciones Web, desde la especificación de requisitos hasta la generación de código.

Sus autores hacen énfasis en que UML es lo suficientemente eficaz para cubrir todos los requisitos que surgen cuando se modelan aplicaciones Web. Para modelar la mayoría de los requisitos utilizan la notación y técnicas de UML estándar, mientras que para modelar los aspectos especiales de las aplicaciones Web tales como navegación y presentación, UWE proporciona un perfil UML especifico. Dicho perfil se define usando los mecanismos de extensión que proporciona el mismo UML: estereotipos, valores etiquetados y restricciones OCL (OMG-OCL, 2011).

La ventaja de usar diagramas UML es su reconocida semántica. Además, la notación y semántica de los elementos de modelado de UML estándar son descritos completamente en la documentación de OMG (OMG, 2010). Igualmente no es difícil para un diseñador de software con conocimientos de UML entender modelos basados en un perfil UML, como UWE lo plantea. Finalmente, permite el uso de una herramienta CASE UML genérica, aunque UWE implementa su propia herramienta CASE.

El proceso de desarrollo UWE

El proceso de desarrollo UWE incluye análisis, diseño y generación de aplicaciones Web. Utiliza un enfoque orientado a objetos, iterativo e incremental basado en UML y el Proceso Unificado de Desarrollo de Software (Rumbaugh et al., 2010). Las actividades básicas de modelado son el análisis de requisitos, diseño conceptual, diseño navegacional y diseño de presentación, complementados con el modelado de los procesos de negocio, despliegue y visualización de escenarios Web.

Para describir los requisitos funcionales, UWE usa los modelos de casos de uso (Rumbaugh et al., 2010), los cuales son etiquetados como de navegación o de proceso. De los casos de uso de navegación de derivan clases navegacionales, y de los casos de uso de proceso las clases que representan una actividad o subproceso. En (Koch and Kraus, 2002) se muestra como los requisitos funcionales de una aplicación Web pueden ser completamente especificados a través de modelos de casos de uso. El modelo conceptual es construido teniendo en cuenta los requisitos funcionales capturados con los casos de uso y muestra el conjunto de elementos estáticos del dominio, UWE representa este modelo a través de diagramas de clases UML. El modelo conceptual UWE intenta ignorar tanto como sea posible los aspectos navegacionales, de presentación y de interacción. Estos aspectos son pospuestos hasta las etapas del diseño de los modelos navegacionales y de presentación.

El modelado de aspectos navegacionales y de presentación no son de interés exclusivo de las aplicaciones Web, pero son críticos en este tipo de aplicaciones, puesto que, separar estos modelos del modelo conceptual incrementa la independencia de los dispositivos cliente en las aplicaciones Web.

El modelo navegacional UWE

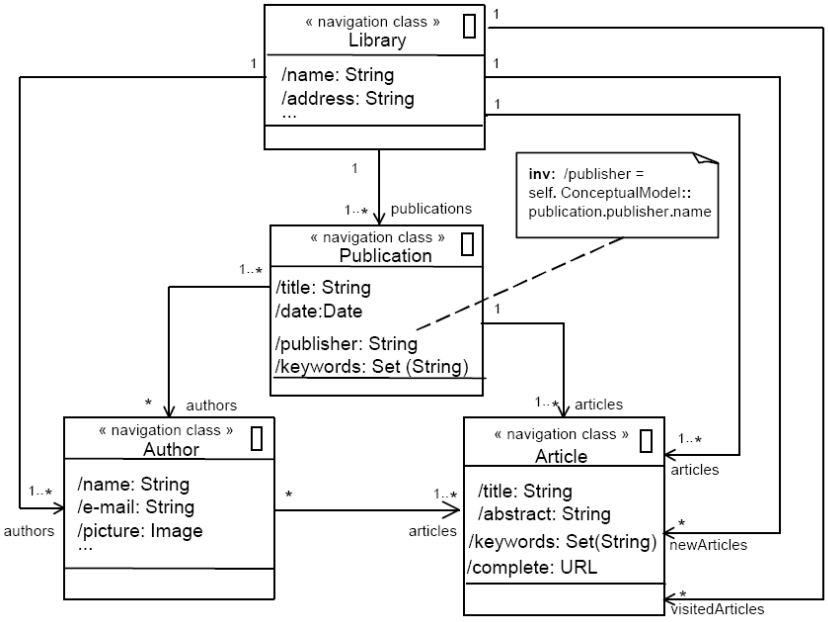
El modelado navegacional de aplicaciones Web en UWE comprende la construcción de dos modelos: el modelo del espacio navegacional y el modelo de la estructura navegacional. El primero especifica que objetos pueden ser visitados por medio de la navegación, este es, por lo tanto, un modelo a nivel de análisis. El segundo define como estos objetos son alcanzados, este es un modelo a nivel de diseño. Estos modelos navegaciones son representados por medio de diagramas de clases estereotipadas.

El modelo del espacio navegacional, como dijimos antes, incluye las clases de los objetos que pueden ser visitados por medio de la navegación y las asociaciones, que definen el camino través del cual los objetos pueden ser alcanzados. UWE proporciona un conjunto de directrices y mecanismos semiautomáticos para el modelado de los aspectos navegaciones partiendo del modelo conceptual (Hennicker and Koch, 2000). La primera decisión a tomar es seleccionar cuales entidades del modelo conceptual deben ser alcanzables por medio de la navegación y a través de que caminos, de forma que se asegure la funcionalidad de la aplicación. Esta decisión de diseño debe estar basada en el modelo conceptual, el modelo de casos de uso y en los requisitos navegacionales que la aplicación debe satisfacer.

Para la construcción del modelo del espacio navegacional se utilizan dos elementos de modelado: *navigation class* y *direct navigability*. *navigation class* modela las entidades que pueden ser alcanzadas mediante la navegación, estas entidades tendrán el mismo nombre que las clases conceptuales y se representarán a través del estereotipo «navigation class». *direct navigability* modela la navegabilidad dirigida desde una clase navegacional fuente a una clase navegacional destino, para determinar la dirección de la navegación, las asociaciones estereotipadas con «direct navigability» deben ser dirigidas (posiblemente bidireccionales). La figura 2-5 muestra un ejemplo de un modelo del espacio navegacional de una aplicación, se asume que todas las asociaciones están implícitamente estereotipadas con «direct navigability». Cabe destacar que solo las clases del modelo conceptual que son relevantes para los aspectos navegacionales son incluidas en este modelo.

El modelo de la estructura navegacional es construido a partir del modelo del espacio navegacional, puede ser considerado como un paso de refinamiento en el proceso de diseño UWE en el cual se agrega al modelo del espacio navegacional elementos de acceso como índices, visitas guiadas, consultas y menús que serán los encargados de desencadenar la navegación.

Figura 2-5. Modelo del espacio navegacional

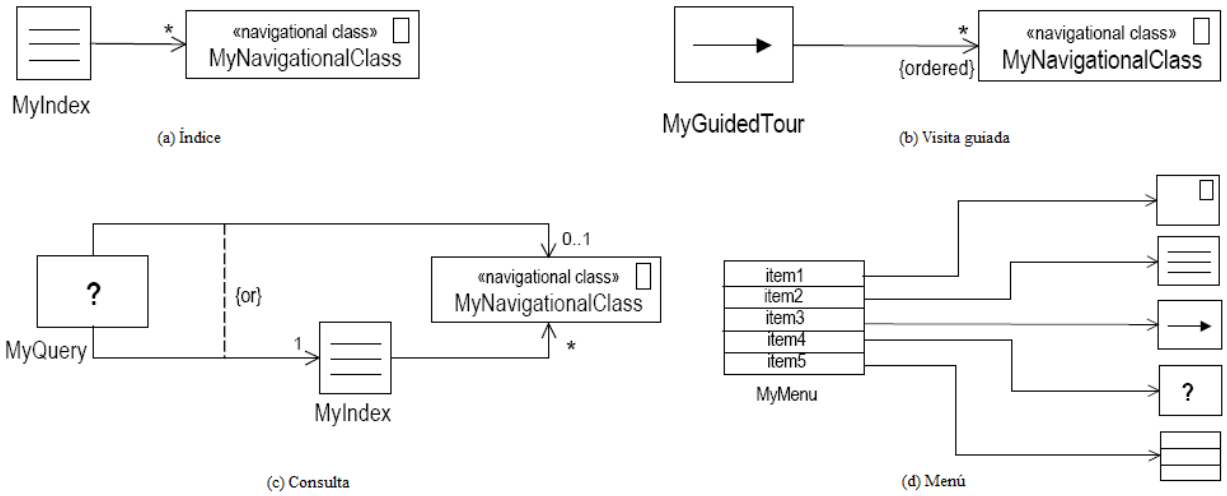


Un índice es un elemento de acceso estereotipado con `«index»` que tiene un nombre y posee un enlace a una instancia de una clase navegacional. Una visita guiada es un elemento estereotipado con `«guidedTour»` que proporciona acceso secuencial a las instancias de una clase navegacional, debe conectarse a una clase navegacional por medio de una asociación dirigida y la cual puede tener la propiedad `{ordered}`. Una consulta es un elemento de acceso estereotipado con `«query»` que posee como atributo una cadena de consulta (puede ser una operación de selección en OCL) que se aplica a las instancias de la clase navegacional a la que esté conectada. Un menú es un elemento de acceso estereotipado con `«menu»` que se compone de un número fijo de ítems, cada item contiene un enlace a una clase navegacional u otro elemento de acceso. En la figura 2-6 se muestra los iconos empleados en UWE para cada uno de los elementos de acceso mencionados y en la figura 2-7 se muestra el modelo de la estructura navegacional basado en el modelo del espacio navegacional de la figura 2-5, como puede verse este tipo de diagrama resulta muy útil en la representación estática de la estructura navegacional.

El modelo de la estructura navegacional muestra como navegar a través del modelo del espacio navegacional por medio del uso de elementos de acceso. El siguiente paso en el proceso de desarrollo UWE es describir como la información del modelo de la estructura navegacional es presentada al usuario. Esto es hecho a través de la construcción del modelo de la presentación, el

cual muestra el diseño de las interfaces de forma abstracta, enfocándose en la organización estructural de la presentación y no en la apariencia física.

Figura 2-6. Elementos de acceso UWE



El modelo de la presentación UWE

El modelo de la presentación es construido a través de una forma particular de diagramas de clases. Estos diagramas de clases utilizan la notación de composición para clases de UML, por ejemplo un contenedor es representado por el elemento de alto nivel estereotipado con `«frameset»`, el cual puede estar compuesto por otros elementos `«frameset»` anidados y por instancias de clases del modelo de la presentación, las cuales son estereotipadas con `«presentational class»`.

Las clases del modelo de la presentación están compuestas por elementos de presentación básicos, tales como anclas, entradas de texto, imágenes, audio y botones. Para construir el modelo de la presentación se tiene que decidir cuales de estos elementos se utilizaran para la presentación de la información de las instancias de las clases navegacionales y para los elementos de acceso.

El modelo de la presentación describe donde y como los objetos que pueden ser alcanzados por medio de la navegación serán presentados al usuario. En UWE, el diseño de la presentación soporta la transformación del modelo de la estructura navegacional en un conjunto

de modelos que muestran la localización estática de los objetos visibles al usuario. En la figura 2-8 se muestra un ejemplo de un modelo de la presentación.

Figura 2-7. Modelo de la estructura navegacional

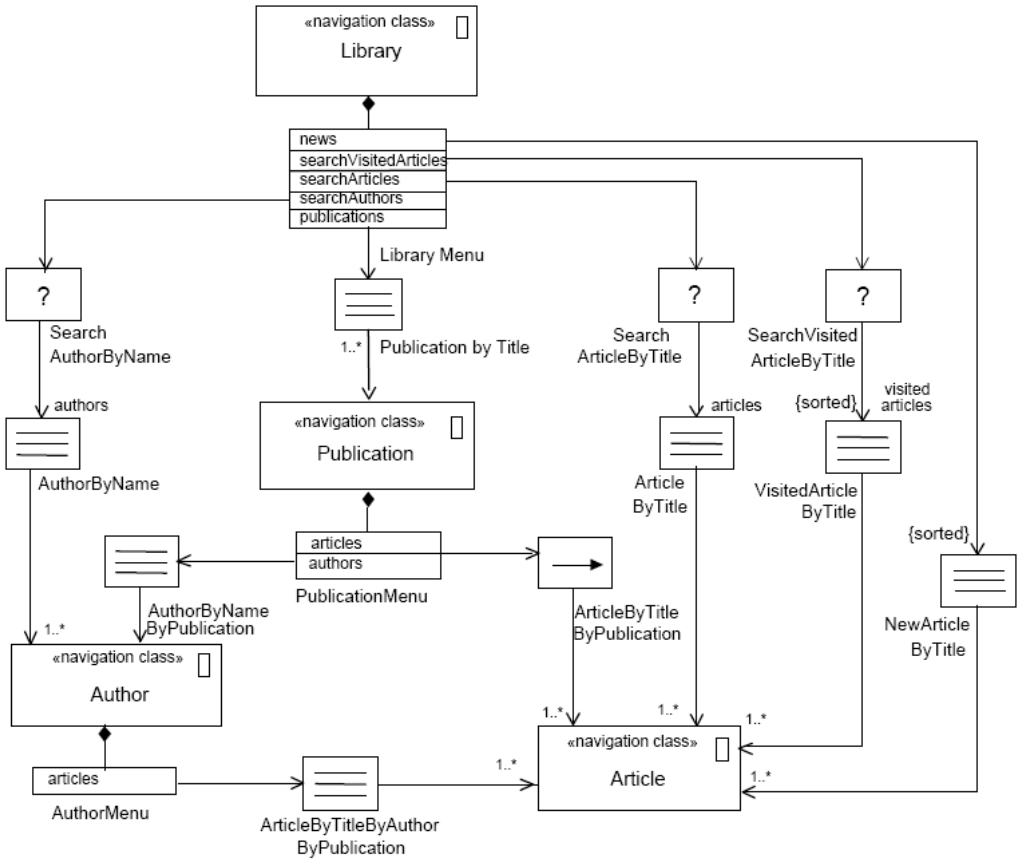
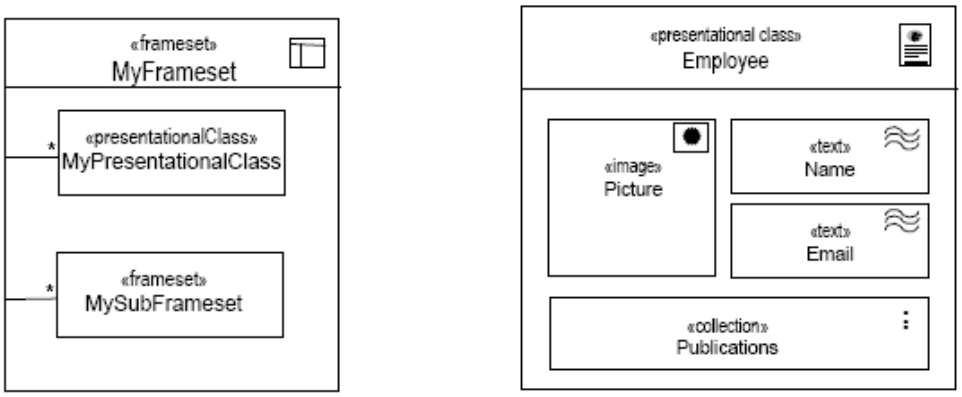


Figura 2-8. Modelo de la presentación



El modelo de procesos de negocio UWE

UWE utiliza Diagramas de Actividad UML para modelar los procesos de negocio de las aplicaciones Web. Los Diagramas de Actividad en general, pueden ser considerados como "*roadmaps*" del comportamiento funcional del sistema.

UWE utiliza casos de uso normales UML para describir procesos de negocio. Estos casos de uso son especificados utilizando diagramas de actividades en la etapa de análisis. Posteriormente, en la etapa de diseño, a estos casos de uso se les asocia clases de proceso, que forman parte del modelo navegacional. A su vez, esas clases de proceso pueden utilizar otras clases de proceso, y especifican su flujo de invocación de mensajes utilizando diagramas de actividades con nodos de llamada.

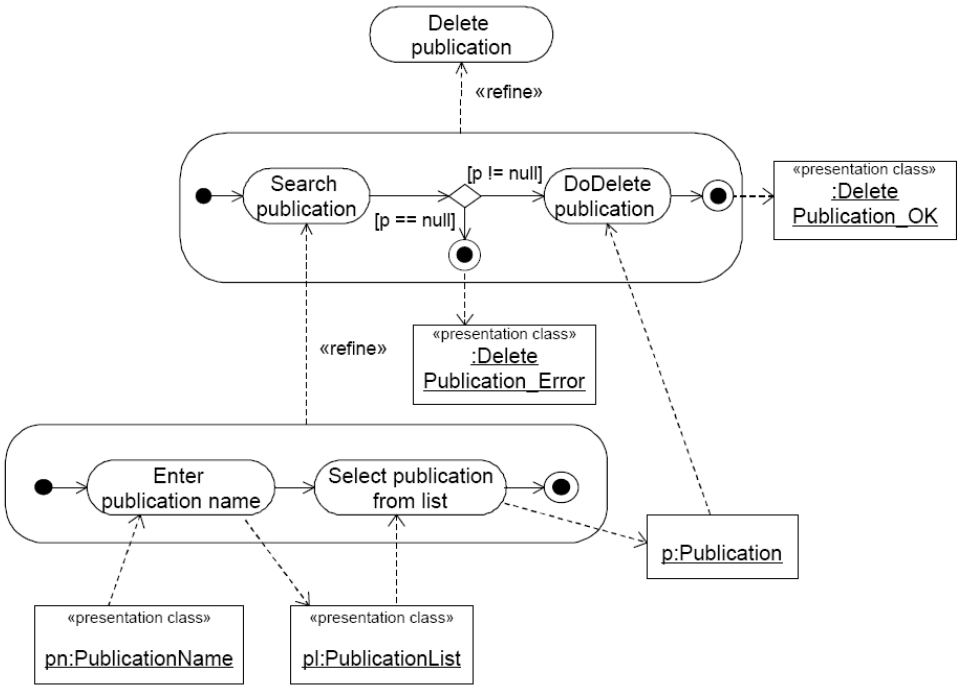
En el modelo de cada proceso de negocio se utiliza una dependencia UML estereotipada «refine» entre la mención del proceso y su Diagrama de Actividad para indicar un refinamiento en el grado de abstracción. Igualmente, dentro del Diagrama de Actividad se distribuyen las sub procesos verticalmente desde los procesos de grano grueso hasta los procesos de grano fino, representando así una jerarquía de procesos. El orden temporal de los procesos es expresado por transiciones entre actividades. El modelado de procesos también comprende la descripción de los objetos que el usuario percibirá. En UWE estos objetos pertenecen al modelo conceptual y al modelo de presentación, la relación entre los procesos y estos objetos es expresada como flujo de objetos visualizados como líneas dirigidas discontinuas. UWE utiliza objetos de presentación entrantes para expresar información entrante desde los usuarios a través de los objetos del modelo de la presentación y objetos de presentación salientes para expresar información saliente hacia los usuarios a través de los objetos del modelo de la presentación. Además, se puede utilizar objetos del modelo conceptual para expresar información de entrada y salida desde y hacia los procesos. La figura 2-9 muestra el Diagrama de Actividad para un proceso llamado "borrar publicación".

Proceso MDD de UWE

En (Koch, 2006) se define el proceso de desarrollo UWE desde el punto de vista de MDD (Bezivin, 2004) haciendo énfasis en la definición de sus metamodelos y en las reglas de transformación de sus modelos. Argumentando que la ingeniería Web es un dominio concreto donde MDD puede ser crítico, particularmente para enfrentar los problemas de evolución y

adaptación a las continuamente cambiantes plataformas y tecnologías Web, se propone basar el proceso de desarrollo UWE en MDA y en otros estándares de OMG (OMG, 2010) como XMI (OMG-XMI, 2007), MOF (OMG-MOF, 2009), OCL (OMG-OCL, 2011) y QVT (OMG-QVT, 2009).

Figura 2-9. Transformaciones MDD en el proceso UWE



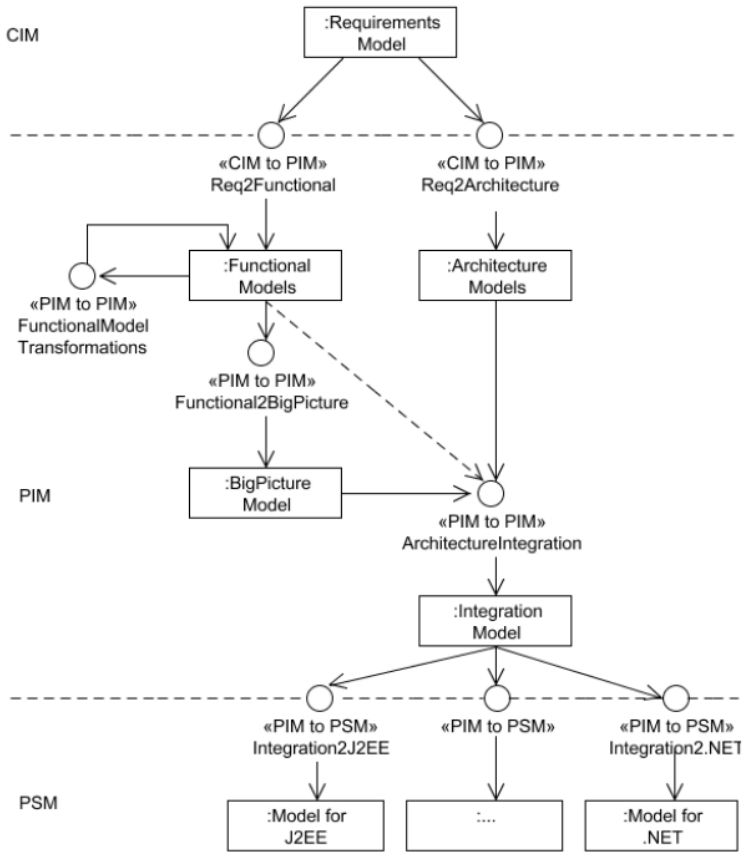
MDA sugiere construir Modelos Independientes de la Computación (CIM), Modelos Independientes de la Plataforma (PIM) y Modelos Específicos de la Plataforma (PSM) correspondientes a diferentes niveles de abstracción. Como se muestra en la figura 2-10, se utiliza esta clasificación para los modelos UWE, el proceso es representado como un diagrama de actividades UML estereotipado. Los modelos son representados como los estados de objetos y las transformaciones son actividades estereotipadas (iconos circulares). Una cadena de transformaciones entonces define un flujo de control.

El proceso comienza en el nivel del modelo de negocio (CIM) definido por el modelo de requisitos UWE. Los modelos de diseño independientes de la plataforma (PIM) son derivados desde estos requisitos. El conjunto de modelos funcionales representan los diferentes aspectos UWE de las aplicaciones Web como el contenido, la navegabilidad, la lógica de negocio, la

presentación y la adaptación de la aplicación Web. Los modelos funcionales son integrados dentro del modelo llamado “BigPicture” en la figura 2-10. La mezcla de los modelos funcionales UWE con el modelo de las características arquitectónicas resulta en un modelo PIM llamado modelo de integración que cubre todos los aspectos funcionales y arquitectónicos. Finalmente los PSM son derivados desde el modelo de integración y desde los cuales el código fuente puede ser generado. El objetivo de este proceso MDD es la transformación automática de modelos en cada paso basado en reglas de transformación.

Para el momento de la publicación del artículo, el mayor problema a el que los autores se enfrentan es encontrar una herramienta que soporte la definición de los metamodelos y la transformación automática de modelos.

Figura 2-10. Transformaciones MDD en el proceso UWE



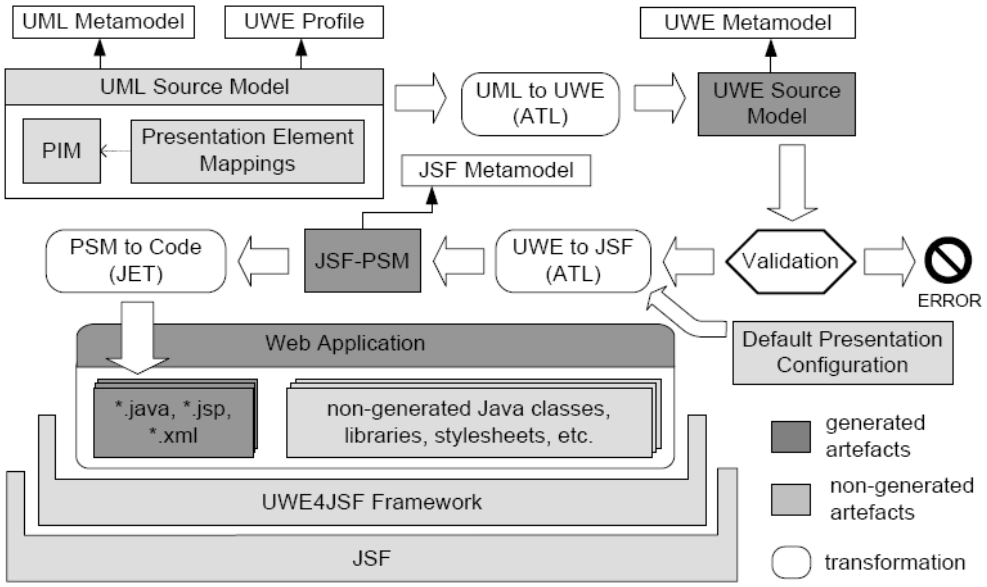
Herramienta CASE UWE4JSF

En (Kroiss et al., 2009) se presenta UWE4JSF, una herramienta CASE que intenta solucionar los obstáculos que presenta el desarrollo dirigido por modelos (MDD) de aplicaciones Web diseñadas con UWE; Las principales características son:

- UWE4JSF se integra completamente en el IDE de Eclipse (Eclipse, 2009), usando, por lo tanto, las tecnologías de transformación basadas en Eclipse; se ha implementado como un conjunto de plugins que dan soporte a la transformación de modelos UWE y a la generación automática de código.
- UWE4JSF tiene la capacidad de generar automáticamente desde los modelos UWE, aplicaciones Web para la plataforma JavaServer Faces (JSF) (Network, 2009a), una tecnología basada en componentes que proporciona un mecanismo flexible y poderoso para la implementación de interfaces de usuario de complejidad arbitraria.
- UWE4JSF hace uso de Object-Graph Navigation Language (OGNL) (OpenSymphony), que es un lenguaje de expresión para Java (Network, 2007) de código libre.
- La generación de la interfaz de usuario está basada en una versión revisada del metamodelo de presentación de UWE.
- UWE4JSF proporciona una forma de generar automáticamente aplicaciones Web de manera intuitiva, con opciones de depuración y alto rendimiento, proporciona mecanismos de validación de los modelos por medio de restricciones especificadas en OCL.

UWE4JSF usa Eclipse Modeling Framework (EMF) (Eclipse, 2009) para almacenar los metamodelos y modelos y permitir el intercambio de estos con otras herramientas CASE UML usando el formato de datos ampliamente soportado EMF-UML. Las transformaciones modelo a modelo (M2M) se realizan a través de Atlas Transformation Language (ATL) (Eclipse), mientras que las transformaciones modelo a texto (M2T) se realizan a través de JET (Eclipse). Estas dos tecnologías basadas en Eclipse se combinan en una cadena de transformaciones como se muestra en la figura 2-11.

Figura 2-11. Proceso de generación de UWE4JSF



El proceso ilustrado en la figura 2-11 comienza con un modelo UML extendido con el perfil UWE compuesto por el modelo independiente de la plataforma (PIM) y el modelo de la presentación. La primera transformación M2M convierte el modelo de entrada a un modelo instancia del metamodelo UWE, el cual es validado usando un conjunto de restricciones OCL. Si la validación es satisfactoria, la próxima transformación genera un modelo específico de la plataforma (PSM) procesando el modelo fuente UWE junto con un modelo de entrada adicional que contiene la configuración por defecto de los elementos de interfaz de usuario del modelo de la presentación. Este PSM es finalmente usado como entrada de la transformación M2T que genera el código fuente de la aplicación, el cual consiste de clases Java y ficheros de configuración. Estos artefactos generados se construyen sobre una plataforma intermedia, llamado framework UWE4JSF, que se ubica en la cima de JSF y ha sido diseñada para reducir la complejidad de la generación de código y las reglas de transformación. Finalmente, la aplicación puede ser ampliada con clases Java no generadas en el proceso.

Herramienta CASE MagicUWE

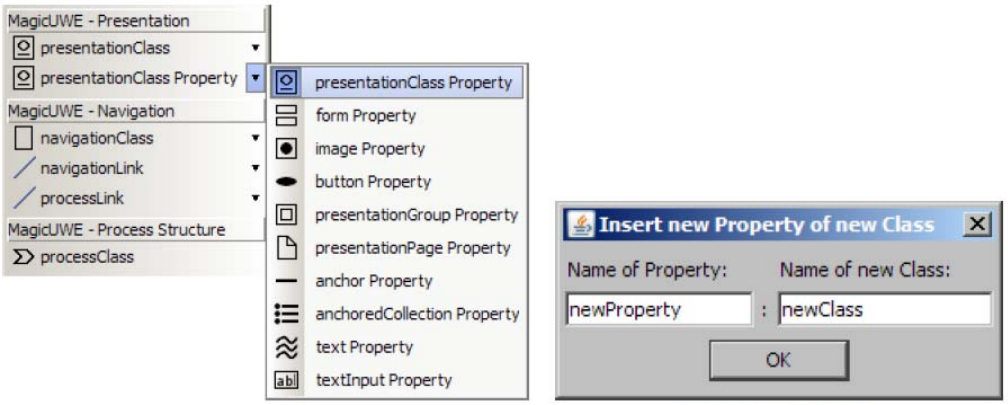
En (Busch and Koch, 2009) se presenta la herramienta MagicUWE, diseñada para proporcionar mayor asistencia a los diseñadores en las actividades de modelado usando el perfil UWE. Implementada como plugin para la herramienta CASE MagicDraw, el desarrollo de

MagicUWE se centró en mejorar la usabilidad, adaptabilidad y extensibilidad de la herramienta para el desarrollo con el proceso UWE.

MagicUWE es fácil de instalar y extender (MagicUWE, 2010), la usabilidad es dada principalmente a través de iconos intuitivos, diferentes tipos de menús y recomendaciones de ayuda. Algunas de sus características son:

- Extiende la barra de herramientas para el uso confortable de los elementos del perfil UWE, incluyendo accesos directos a ellos.
- Menús específicos para crear cada uno de los diferentes diagramas UWE (contenido, navegacional, presentación, etc.) como se muestra en la figura 2-12 a.
- Menús contextuales para editar propiedades de los diagramas y/o de sus componentes, como se muestra en la figura 2-12 b.

Figura 2-12. Creación y edición de propiedades con MagicUWE



(a) Menú para crear diagramas y componentes UWE

(b) Edición de propiedades UWE

Para enfrentar las nuevas características de UWE y las próximas versiones de MagicDraw, MagicUWE se ha diseñado de una forma muy modular, con extensa documentación y con capacidad de obtener y ejecutar nuevas versiones del plugin automáticamente. De esta forma la herramienta es fácilmente extensible y adaptable, lo cual es un requisito indispensable en el continuo proceso de mejoramiento de las herramientas CASE.

2.1.3 Web Modeling Language – WebML

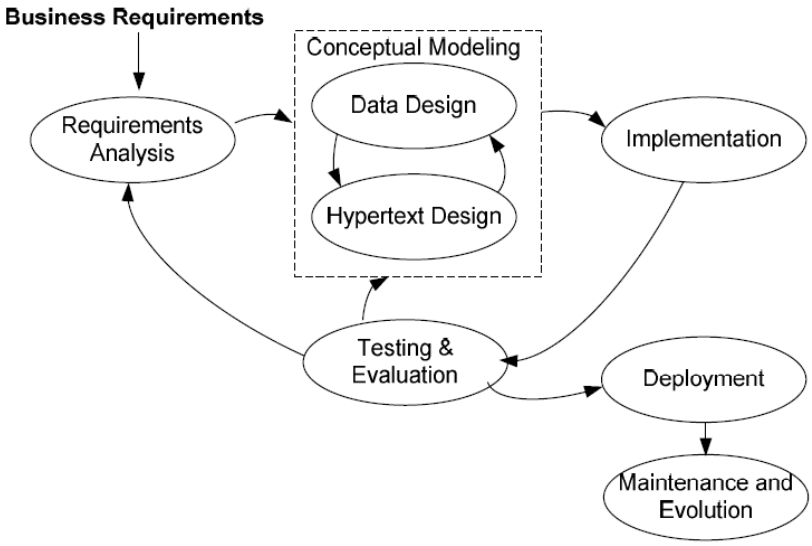
Web Modeling Language – WebML (Ceri et al., 2000) es un lenguaje de modelado para el diseño de aplicaciones que hacen uso intensivo de datos y que serán desplegadas en la Web o sobre arquitecturas orientadas a servicios utilizando servicios Web. WebML permite a los diseñadores expresar las características de las aplicaciones Web a alto nivel sin tener en cuenta los detalles arquitectónicos. Los conceptos WebML son representados a través de modelos gráficos intuitivos, los cuales son a la vez, fácilmente soportados por herramientas CASE y entendibles para los miembros no técnicos del equipo de desarrollo. Además, WebML utiliza XML (W3C, 2011a) para representar los modelos generados en cada etapa del desarrollo, lo que permite que sean fácilmente consumidos por generadores de código fuente para producir automáticamente la implementación de las aplicaciones Web. De todas formas, WebML cuenta con su propia herramienta de desarrollo llamada WebRatio (WebRatio, 2011) que permite construir sus modelos y generar el código de la aplicación Web mediante la aplicación de transformaciones XSLT (W3C, 1999).

Proceso de desarrollo WebML

El proceso de desarrollo WebML consiste en diferentes etapas que se aplican de forma iterativa e incremental, en cada iteración la aplicación es evaluada y probada, y de acuerdo a los resultados puede ser extendida o modificada para cumplir los nuevos o cambiantes requisitos. La figura 2-13muestra las etapas en el proceso de desarrollo WebML, la etapa de *modelado conceptual* que tiene especial interés en nuestro trabajo, está compuesta por el *diseño del modelo de datos* y el *diseño del hipertexto* a un alto nivel de abstracción e independientemente de los detalles de implementación.

El *diseño del modelo de datos* consiste en organizar la información identificada previamente en la etapa de análisis de requisitos dentro de un modelo de datos coherente y comprensivo posiblemente complementado a través de objetos secundarios. El *diseño del hipertexto* produce esquemas *site view* en la cima de modelo de datos anteriormente definido. Los *site view* muestran la composición del contenido dentro de las páginas así como la navegación y la interconexión de componentes.

Figura 2-13 Etapas en el proceso de desarrollo WebML



Como lo detallaremos más adelante, las etapas del proceso de desarrollo WebML se centran en la construcción de cuatro modelos: modelo estructural, modelo de hipertexto, modelo de presentación y modelo de personalización. La "separación de problemas" es un requisito fundamental para cualquier lenguaje de modelado Web. En WebML este requisito se satisface asignando diferentes clases de especialistas a la construcción de cada uno de sus modelos: el experto en modelado de datos diseña el modelo estructural, el arquitecto de la aplicación diseña las páginas y la navegación entre ellas (modelo de hipertexto), el diseñador gráfico diseña los estilos de presentación de las páginas (modelo de presentación) y el administrador de la aplicación diseña las opciones de personalización (modelo de personalización).

El modelo estructural

Para la especificación del modelo de datos subyacente a las aplicaciones Web, llamado modelo estructural, WebML hace uso de notaciones clásicas, como la notación del modelo Entidad-Relación E/R (Pin-Shan Chen, 1976a) ó diagrama de clases UML (OMG-UML, 2010). Los elementos fundamentales del modelo estructural son las entidades, definidas como contenedores de elementos de datos, y las relaciones, definidas como conexiones entre entidades. Las entidades tienen propiedades con un tipo asociado, pueden estar organizadas jerárquicamente y las asociaciones pueden estar restringidas por medio de restricciones de cardinalidad.

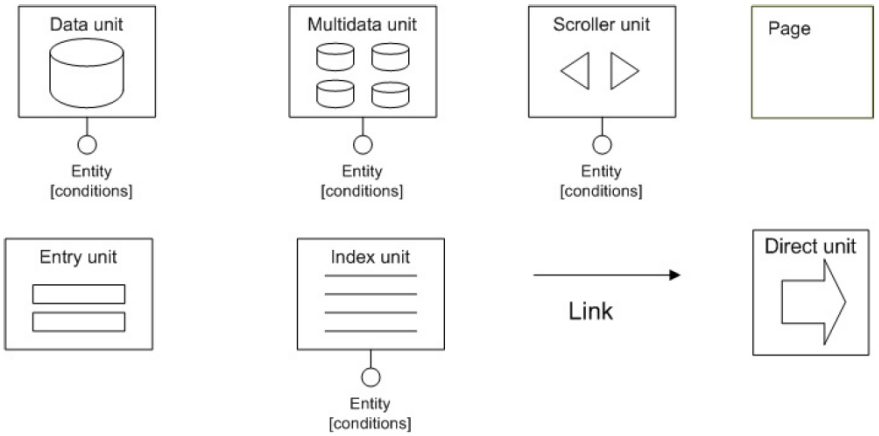
El modelo de hipertexto

El modelo de hipertexto describe las vistas del modelo estructural que estarán publicadas en el sitio Web. Se divide en dos modelos, el modelo de composición y el modelo de navegación.

El *modelo de composición* define qué páginas estarán contenidas en el sitio Web, y qué contenido tiene cada página. Los elementos del modelo de composición son llamados *unidades* y están inspiradas en las primitivas de acceso RMM (Isakowitz et al., 1995). Las unidades para construir el modelo de composición se muestran el la figura 2-14 se detallan a continuación:

- La unidad de datos *(data unit)*: representa una entidad del modelo estructural y sus atributos visibles. Es decir, una unidad de datos está definida por una entidad del modelo relacional y los atributos que debe mostrar. La unidad multidatos *(multidata unit)* presenta varias instancias de una entidad simultáneamente, de acuerdo a algún criterio de selección.

Figura 2-14 Notaciones para Unidades del modelo de composición



- Un índice *(index unit)*: representa a un conjunto de referencias a entidades, donde cada referencia presenta una descripción de la entidad a la que apunta. Pueden ser vistos como una lista de objetos (entidades o instancias de componentes), sin presentar información detallada de cada objeto.
- Una barra de desplazamiento *(scroller unit)*: representa la visualización secuencial de un conjunto de entidades y está asociado a una unidad de datos; es más conocida como visita

guiada. Muestra comandos para accesar a los elementos del conjunto ordenado de objetos (primero, último, anterior, siguiente, etc.).

- Una unidad de filtrado *(entry unit)*: está asociada a una entidad y contiene una lista de atributos de filtrado y un predicado condicional. Es un formulario que permite establecer condiciones de selección a índices, barras de desplazamiento ó unidades multidatos.
- Una unidad directa *(direct unit)*: representa una relación uno a uno entre dos entidades. No muestra información, son usadas para denotar la conexión a un objeto simple que está semánticamente relacionado con otro objeto.
- Una Página *(page)*: es una abstracción de una región autocontenida de la interfaz de usuario. La granularidad de las unidades anteriores puede ser demasiado fina para los requisitos de composición de una aplicación, los cuales normalmente demandan que la información contenida en algunas entidades sea entregada junta. Para cumplir con este requisito, WebML proporciona la noción de Página.

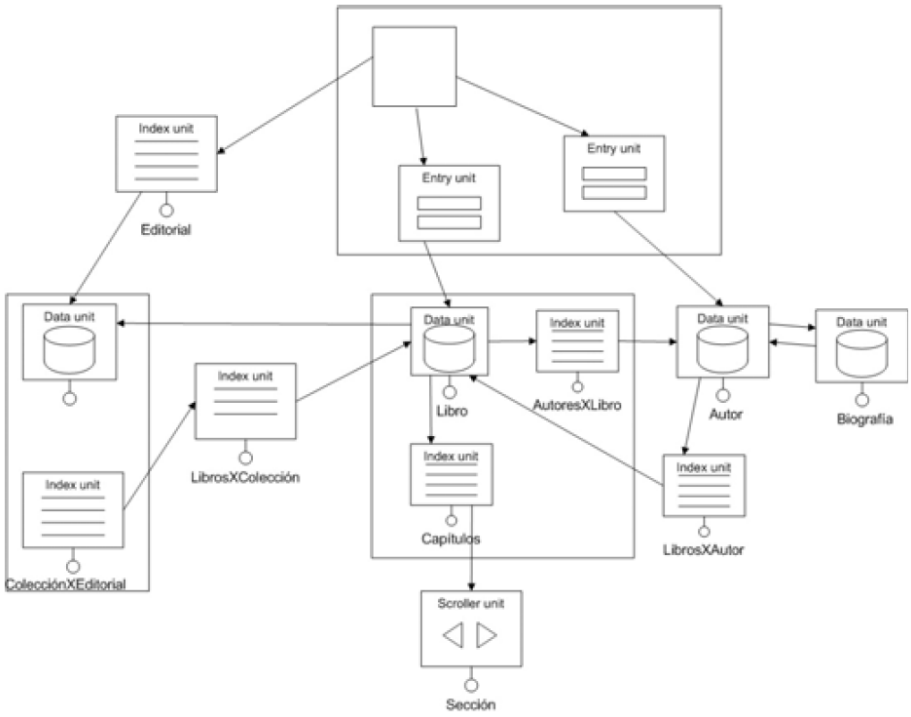
El *modelo de navegación* especifica la forma en la cual las unidades y las páginas son enlazadas. Unidades y páginas no pueden existir aisladamente, deben conectarse de alguna forma para producir estructuras navegacionales. Para este propósito, WebML proporciona la noción de enlace *(link)*. Los enlaces pueden ser de dos clases, *contextuales* cuando relacionan a dos unidades, este tipo de enlace lleva alguna información (llamada contexto) desde la unidad fuente a la unidad destino, el contexto es utilizado para determinar el objeto o conjunto de objetos que serán mostrados en la unidad destino. *No-contextuales* cuando relacionan páginas de forma arbitraria, independientemente de las unidades que contengan y de la relación semántica entre éstas unidades y el modelo estructural.

En WebML, unidades, páginas y enlaces pueden expresar estructuras navegacionales de complejidad arbitraria, donde pueden existir múltiples páginas intermedias para soportar la navegación hacia una unidad de datos. Estas configuraciones son llamadas “cadenas navegacionales”. Las cadenas navegacionales frecuentemente usadas son conocidas como “patrones Web WebML”. En (Bernstein, 1998) se presenta una selección de patrones Web WebML que describen una gran variedad de situaciones que ocurren el la práctica.

En la figura 2-15 se muestra un ejemplo de un modelo de hipertexto que contiene los modelos de composición y de navegación. El modelo de composición está formado por cada

nodo individual y por cada página. La página principal muestra un enlace a una página que es un índice de *editoriales* y contiene dos formularios de búsqueda. En WebML una unidad de datos es una selección de atributos de una entidad de datos del modelo estructural; por ejemplo, la unidad de datos *biografía* se deriva de la entidad *autor* y sólo selecciona el campo *biografía*. El modelo de navegación está integrado por los enlaces que cruzan la frontera de las páginas y que salen de las unidades que se presentan de forma individual.

Figura 2-15 Modelo de hipertexto



El modelo de presentación

El modelo de la presentación se ocupa de modelar el *look and feel* de las páginas identificadas en el modelo de la composición. La presentación de las páginas WebML es construida a través de hojas de estilos. Una hoja de estilos establece la apariencia gráfica de las páginas y de los elementos contenidos en ellas y, además, es independiente del lenguaje usado para construir las páginas. En WebML, la forma más común de construir el modelo de presentación es a través de la aplicación de hojas de estilo XSL (W3C) a los documentos XML

que representan las instancias del modelo navegacional, siendo el resultado de la transformación un documento HTML (W3C, 2011b).

El modelo de personalización

El modelo de la personalización está formado por las entidades predefinidas "usuario" y "grupo", que son explícitamente modeladas en el modelo estructural. Las características de estas entidades se usan para guardar y mostrar tanto contenido individualizado como apariencia gráfica personalizada, por ejemplo, sugerencias de compra, listas de favoritos y recursos para personalizaciones gráficas.

El modelo de procesos de negocio con WebML

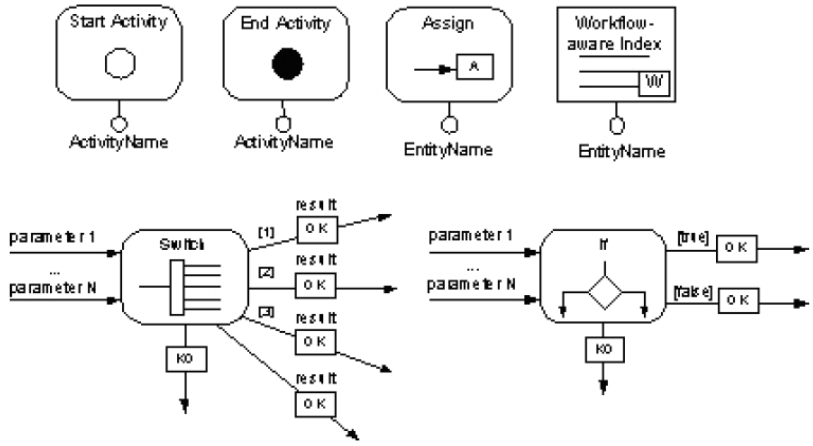
Como lo comentamos anteriormente, WebML fue diseñado para modelar aplicaciones Web que hacen uso intensivo de datos, sin embargo, dado que las aplicaciones Web han evolucionado convirtiéndose en aplicaciones basadas en procesos de negocio, WebML ha sido extendido para modelar este tipo de aplicaciones, aportándoles además los beneficios del modelado conceptual de alto nivel y la generación de código automático.

Integrar procesos de negocio en el modelado de aplicaciones Web significa permitir que las interfaces Web puedan lanzar procesos de negocio e incorporar restricciones de negocio en los modelos de navegación. Para cubrir estos aspectos, WebML ha sido extendido de la siguiente manera:

- Se ha agregado un *modelo de procesos de negocio*, consistente en un nuevo diagrama de flujo de trabajo que representa los procesos de negocio que serán ejecutados en términos de actividades con restricciones de precedencia y los actores/roles encargados de ejecutar cada actividad.
- El *modelo estructural* es extendido con un conjunto de objetos (entidades y relaciones) necesarios para guardar la información que pueda requerir la ejecución de los procesos de negocio.
- El *modelo de hipertexto* es extendido para especificar las actividades de negocio que se pueden lanzar desde las interfaces de usuario y los enlaces navegacionales dependientes del flujo de trabajo.

Para modelar los procesos de negocio, WebML ha adoptado la notación *Business Process Modelling Notation (BPMN)* (Brambilla et al., 2006). El modelo de hipertexto ha sido extendido con las primitivas que se muestran el la figura 2-16: *"start activity"*, *"end activity"*, *"assign"*, *"switch"* e *"if"*.

Figura 2-16. Primitivas de proceso de WebML



"Start Activity" y *"end Activity"* delimitan una porción del modelo de hipertexto dedicado a la ejecución de una actividad. La primitiva *"start activity"* implica la creación de una instancia de la actividad y el registro del momento de creación. En cambio, la primitiva *"end activity"* asigna a la actividad el estado de completada y registra el momento en que ocurre ello. La primitiva *"assign"* permite asignar un dato generado por una actividad a otra que lo consume. Las primitivas *"start activity"*, *"end activity"* y *"assign"* son de control y no tienen un efecto en el modelo de navegación. Las primitivas *"switch"* e *"if"* efectúan una navegación condicional basada en el estado del proceso; ambas repercuten en el modelo de navegación porque afectan en la decisión del nodo destino de una navegación.

Para integrar el modelo de proceso de negocio y el modelo de hipertexto, el diseñador WebML, a partir de las actividades modeladas en BPMN tiene que diseñar un modelo de hipertexto equivalente, con las primitivas mostradas anteriormente.

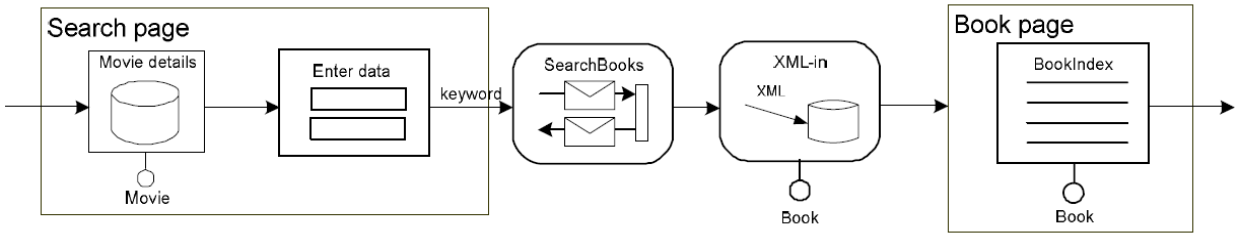
El modelo de interacción con servicios Web en WebML

Como comentamos al principio, WebML permite modelar aplicaciones que serán desplegadas sobre arquitecturas orientadas a servicios utilizando servicios Web. Para modelar las interacciones con servicios Web, WebML cuenta con un conjunto de unidades especiales llamadas *"Web Service Units"* (Manolescu et al., 2005) que representan interfaces WSDL (W3C) con las operaciones de los servicios Web.

WebML soporta las cuatro categorías de operaciones propuestas en WSDL: *"one-way operations"*, *"request-response operations"*, *"notification operations"*, *"solicit and response operations"*.

La figura 2-17 muestra un ejemplo de modelado de una operación *"request-response"* llamada "SearchBooks" que permite obtener una lista de libros desde un servicio Web cuyos títulos coincidan con un criterio de búsqueda.

Figura 2-17. Modelado de una operación *"request-response"* en WebML



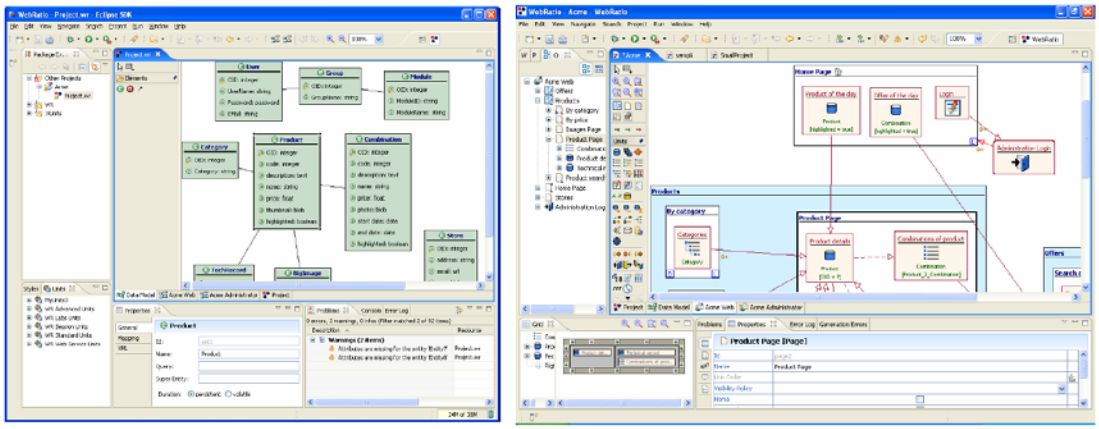
De acuerdo al modelo, un usuario puede navegar hasta la página "Search page", donde una unidad de entrada acepta un criterio de búsqueda. Con esta información, se compone un mensaje de petición y se envía a la operación "SearchBooks" del servicio Web. El usuario espera por el mensaje de respuesta que contiene la lista de libros que satisfacen el criterio de búsqueda. El conjunto de instancias de la entidad "Book" son creadas a través de la unidad XML-in (la cual recibe como entrada datos XML y los transforma en datos relacionales) y mostrados a través de la unidad "BookIndex".

WebRatio: Una herramienta para modelar con WebML

WebRatio 5 es una herramienta implementada como un conjunto de plugins para Eclipse (Eclipse, 2009) orientado hacia el cubrimiento completo del proceso de desarrollo WebML, para

ello soporta completamente los metamodelos de WebML incluyendo las extensiones para el modelado de procesos de negocio y el modelado de interacción con servicios Web. WebRatio 5 proporciona una interfaz gráfica para el diseño de aplicaciones con WebML que está compuesta por un conjunto de editores de diagramas, transformadores y validadores de modelos. La figura 2-18 (a) muestra el editor de diagramas para el diseño del modelo estructural WebML y la figura 2-18 (b) muestra el editor de diagramas para el diseño del modelo de hipertexto WebML.

Figura 2-18. Editores para modelo estructural y de hipertexto en WebRatio



En WebRatio, los modelos WebML son transformados siguiendo la metodología *Model Driven Engineering* (MDE) para incrementar la productividad en el proceso de desarrollo. En particular, las transformaciones tienen como objetivo la generación de código y la documentación automática de los proyectos. Los generadores de código de WebRatio, partiendo de los modelos WebML, producen aplicaciones Web que se ejecutan sobre la plataforma J2EE (Network, 2007). Para generar documentación, WebRatio permite que los diseñadores especifiquen un conjunto de propiedades descriptivas y tiene la capacidad de automáticamente generar documentación en diferentes formatos (PDF, HTML, RTF, etc.) y con diferentes estilos gráficos.

La arquitectura de WebRatio es completamente extensible. Además de las primitivas y patrones WebML existentes que deberían cubrir gran parte de los requisitos de las aplicaciones Web, WebRatio permite especificar nuevos componentes (por ejemplo, nuevas unidades WebML) e incluirlos en el diseño de la aplicación y en el framework generador de código.

2.1.4 IBM – Rational Software Architect

A pesar de que IBM-Rational Software Architect, IBM-RSA (IBM, 2010), es básicamente una herramienta CASE para UML, también ofrece un poderoso entorno integrado y configurable para el diseño y desarrollo de software que ayuda a gestionar la arquitectura y diseñar y administrar la evolución de las soluciones a todos los miembros de un equipo implicados en el desarrollo de un sistema software. Las características de modelado y edición visual están diseñadas para mejorar la productividad, fortalecer el control arquitectónico y facilitar la experiencia de diseño de código. IBM-RSA asiste al desarrollo de aplicaciones usando modernas tecnologías de la industria del software incluyendo J2EE (Network, 2009b) y tecnologías de servicios Web, da soporte a la iniciativa MDA (OMG-MDA, 2003) de OMG (OMG, 2010), soporte a los estándares asociados al estilo de desarrollo SOA (Kumar et al., 2009) y soporte a las capacidades JavaServer Faces (Network, 2009a) que habilitan el desarrollo rápido de aplicaciones.

En esta sección se mostrarán las características que ofrece IBM-RSA para el desarrollo de aplicaciones Web utilizando el enfoque estandarizado ofrecido por Java EE (Network, 2007), el cual contribuye a facilitar el desarrollo y despliegue de las aplicaciones Web. También se comentará el soporte proporcionado por IBM-RSA para el desarrollo dirigido por modelos (MDD) (Hailpern and Tarr, 2006), el cual está basado en el framework Eclipse (Eclipse, 2009), pero IBM-RSA va más allá, en lugar de simplemente integrar Eclipse, ha construido nuevas capacidades MDD en la cima de este framework formando una herramienta MDD más completa (Swithibank et al., 2005).

Las aplicaciones Web pueden cubrir un amplio espectro de funcionalidades, como soporte para transacciones online, entornos de trabajo colaborativo y aplicaciones Web móviles entre otras. En general, las aplicaciones Web son descritas en (Murthy and Shi, 2003) como aplicaciones que utilizan la Web como la infraestructura de la aplicación para ejecutar su funcionalidad que puede tener una complejidad lógica significativa. Dado que la Web es usada como plataforma para sistemas que pueden cumplir muy diversas funcionalidades, hay una gran variedad de tecnologías disponibles para los desarrolladores, y por supuesto, se requiere un amplio rango de habilidades para el desarrollo de los componentes de las aplicaciones Web: programadores, diseñadores gráficos, diseñadores de páginas Web, desarrolladores de contenido, diseñadores y administradores de bases de datos, expertos en redes y seguridad Web,

administradores de servidores Web, etc. Las responsabilidades de cada componente están implementadas en diferentes capas lógicas, donde cada capa tiene un rol diferente en el sistema. Esto se conoce como arquitectura multicapas (Microsystems, 2007).

Java EE simplifica el desarrollo de aplicaciones proporcionando un conjunto de especificaciones para el desarrollo de aplicaciones multicapa. Estas especificaciones se basan en componentes modulares y estandarizados, y en servicios que hacen uso de estos componentes. De esta forma en la especificación Java EE, el modelo de la aplicación encapsula las capas de funcionalidad en tipos específicos de componentes: la capa de negocio, sobre todo si soporta lógica distribuida, es encapsulada en componentes Enterprise Java Beans (EJB). La capa que interactúa con el cliente puede ser presentada a través de páginas Web HTML planas, Java Servlets ó sú equivalente orientado a la entrada salida, las Java Server Pages (JSPs). Para la capa de acceso a datos se proporciona componentes JDBC, y así para cada una de las capas se especifican diferentes componentes.

Hoy en día el desarrollo de aplicaciones Web puede verse facilitado desde distintos frentes: *frameworks*, generadores de código y entornos de desarrollo integrado (IDE). El objetivo final es incrementar la rapidez y facilidad de desarrollo de las aplicaciones, así como también ayudar a obtener una alta calidad en las mismas. Los *frameworks* proveen un amplio rango de servicios para las aplicaciones, tanto en la etapa de desarrollo como en la etapa de despliegue. Los *framework* pueden brindar servicios específicos como acceso a base de datos, *logging*, *caching*, manejo de errores, monitorizaciones, *testing*, seguridad, etc. También existen frameworks que pueden suplir un rango mucho mayor de funcionalidades hasta proveer la infraestructura para aplicaciones completas, como por ejemplo Oracle-ADF (Oracle, 2010d). Por lo general los frameworks son usados dentro de IDEs y otras herramientas de desarrollo.

Un IDE es un entorno de programación que, como mínimo, consiste de una interfaz gráfica de usuario, un editor de código, un compilador y un *debugger*. Visual Studio .Net y Eclipse son ejemplos de IDE. Algunos IDE con más capacidades que los citados anteriormente, pueden permitir la inclusión de diversos frameworks, generar código automáticamente y soportar especificaciones estandarizadas como las especificaciones Java EE. Una herramienta que tenga estas características debería ser considerada una herramienta muy valiosa.

IBM-RSA es un ejemplo de IDE integrado dentro de una herramienta CASE que da soporte a las especificaciones Java EE, UML 2.0 (OMG-UML, 20110), arquitecturas dirigidas

por modelos (MDA) y promueve la unificación de todos los aspectos relacionados con el diseño y desarrollo de software. Los beneficios de IBM-RSA son el soporte para la inclusión de diferentes frameworks, tal como Java Server Faces (JSF) y Struts (Apache, 2007) entre muchos otros, y asistentes para la creación y despliegue de componentes Java EE; la utilización de IBM-RSA debería permitir al equipo de desarrollo prestarle más atención a los requisitos de negocio que a los detalles de las especificaciones estandarizadas para el desarrollo de las aplicaciones Web de Java EE. Como resultado, el proceso de desarrollo debería ser más productivo y las aplicaciones deberían ser fáciles de mantener.

Para comprender como la utilización de IBM-RSA impacta en el desarrollo de las aplicaciones Web, es necesario examinar la responsabilidad de cada capa y las soluciones tecnológicas que Java EE nos propone para cada una de ellas así como otras soluciones alternativas. Es necesario entender estas tecnologías y sus beneficios, y como estas son soportadas por IBM-RSA. En la figura 2-19 se muestra la interfaz de diseño de aplicaciones Web de IMB-RSA, como puede verse, esta interfaz permite "arrastrar y soltar" componentes JavaServer Faces en el diseñador de páginas, componentes de interfaz de usuario y objetos de servicios de datos.

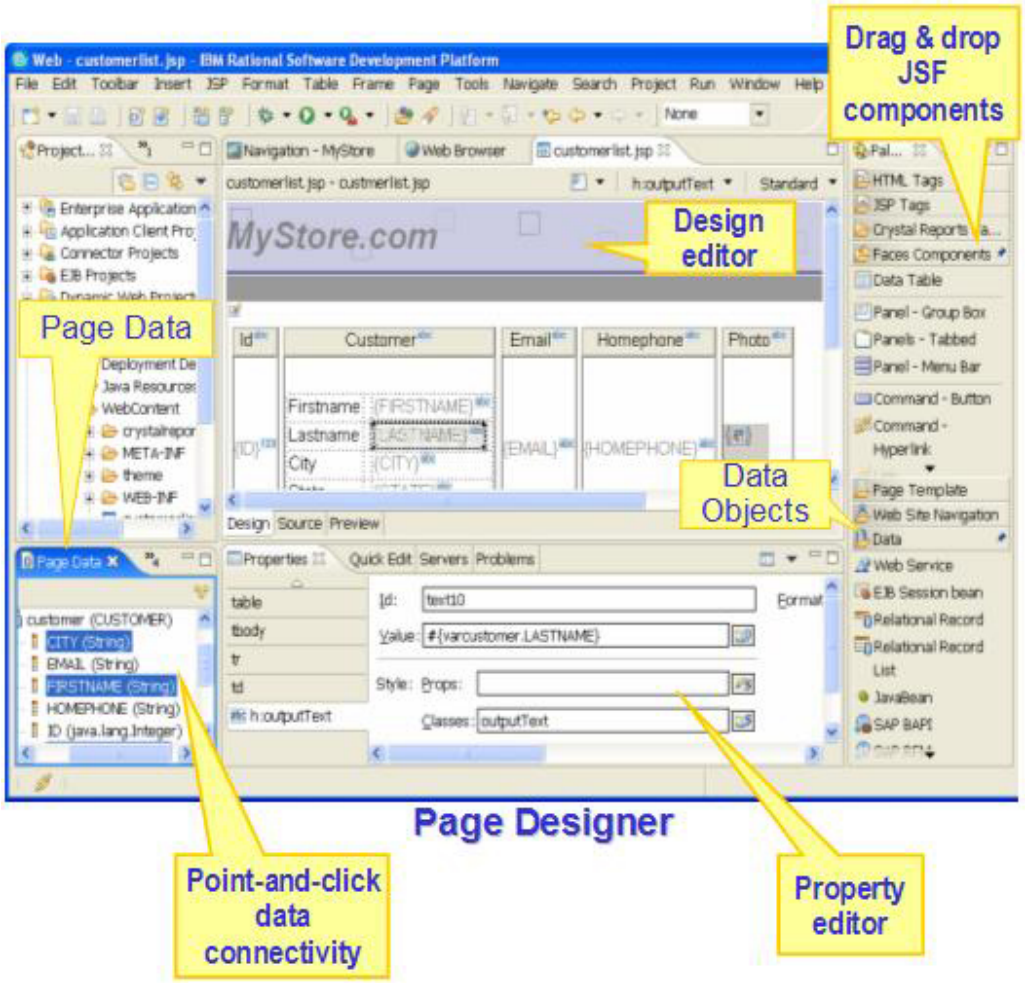
Capa de presentación

La capa de presentación es responsable en cómo los usuarios interactúan con la aplicación, incluyendo la disposición del diseño de la página. Para esto, se cuenta con algunas tecnologías que incluyen HTML, JSP y JavaScript. Estas páginas pueden enlazar con clases procesos computacionales implementados en Java mediante Servlets. Un problema potencial con la tecnología JSP es que puede requerir el conocimiento de diferentes lenguajes como HTML, Java y librerías "*Tag*" como CSS. Otras tecnologías como FreeMarker (Freemarker, 2010), entre otras, han sido desarrolladas con el objetivo de separar el diseño visual de la páginas de la programación lógica de la mismas y para las cuales IBM-RSA también brinda soporte.

Otra solución últimamente más usada y soportada también por IBM-RSA y que separa la lógica de la aplicación de la presentación de las páginas es JavaServer Faces (JSF). JSF es un *framework* para aplicaciones Web que hace parte de las especificaciones J2EE. JSF utiliza un conjunto de API que representan componentes de interfaz de usuario, los cuales soportan controladores de eventos, validación de entradas y navegación de páginas; estos componentes de

interfaz de usuario se enlazan al código de la aplicación sin necesidad de secuencias de comandos adicionales. Pero IBM-RSA no solo soporta JSF, ofrece un componente basado en JSF llamado JWL (JSF *Widget Library*) (Starkey et al., 2008) que implementa y extiende la especificación JSF. JWL hace más fácil crear aplicaciones basadas en JSF.

Figura 2-19. Interfaz de diseño de aplicaciones Web de IBM-RSA



Recientemente, AJAX (Hertel) se ha convertido en una tecnología muy popular debido a que mejora enormemente la experiencia que los usuarios perciben cuando interactúan con una aplicación Web. Sin embargo, es muy difícil desarrollar aplicaciones AJAX sin usar un framework que soporte esta tecnología. JWL proporciona toda la potencialidad de AJAX a través

de los componentes JSF haciendo fácil desarrollar componentes JSF con toda la capacidad que ofrece AJAX.

Capa de negocio

La capa de negocio define la aplicación. Esta interactúa con los datos de la aplicación y modela la lógica del negocio, sirviendo de interfaz entre la capa de presentación y la capa de recursos. Funcionalmente ejecuta la lógica que procesa las peticiones de los clientes y proporciona la información requerida por la capa de presentación. Las tecnologías para esta capa incluyen Enterprise JavaBeans (EJB) y Java Servlets. Los EJBs pueden ser de entidad o de sesión. El rol de un EJB de entidad en una aplicación Web es la administración y persistencia de los datos. Un EJB de sesión es usado para controlar como los datos son procesados. Los Servlets son usados para acceder al estado de los datos desde los EJB y pasar la información a un componente JSP en la capa de presentación.

Los EJB son complejos y difíciles de usar. Un EJB de entidad requiere tres clases: dos interfaces y la clase que las implementa. El despliegue de la aplicación también se dificulta por la necesidad de empaquetar las entidades dentro de un fichero EJB JAR que requiere un descriptor de despliegue. Además una aplicación EJB involucra componentes “ClassLoaders” que pueden causar problemas por la incapacidad de encontrar las clases a cargar o por incoherencia en las versiones.

Para la creación de un proyecto EJB, IBM-RSA proporciona la capacidad para seleccionar la versión EJB y opcionalmente crear el fichero JAR que contiene las interfaces y clases. IBM-RSA tiene la capacidad de crear diagramas UML de los componentes EJB y, a partir de estos generar automáticamente las interfaces locales o remotas. IBM-RSA proporciona también cuatro tipos de clases adicionales llamadas “Acess Beans” para abstraer los EJB: “Java Wrappers”, “EJB Factories”, “Data Classes” y “Copy Helpers”. Estas clases ayudan a simplificar el acceso a datos desde los EJB así como incrementar su rendimiento.

IMB-RSA soporta “Java Data Objects” (JDO) que pueden ser una solución para persistir objetos planos Java. Dependiendo de los requisitos de la aplicación, estos pueden reemplazar a las entidades EJB. Otra tecnología soportada por IBM-RSA de la capa de negocio es Hibernate (Hibernate, 2006), que proporciona capacidades de persistencia y consulta de objetos sin depender de múltiples API. Hibernate puede definirse como una extensión orientada a objetos de

SQL, lo cual la convierte en una herramienta poderosa para los desarrolladores familiarizados con SQL.

Capa de recursos

La capa de recursos contiene los datos del negocio y la información del estado de la aplicación. Su responsabilidad es proporcionar acceso a otras tecnologías que soportan los servicios requeridos por los componentes. Java EE ofrece un número de servicios que están disponibles a través de API, estos servidores de aplicaciones incluyen "*Java Naming and Directory Interface*" (JNDI) que proporciona a los componentes Java EE la capacidad de buscar las interfaces usadas para crear EJB y conexiones JDBC; este servicio habilita a las aplicaciones a ejecutar operaciones de directorio estándar asociando nombres a objetos y retornando objetos basados en un nombre suministrado.

"*Java Data Base Connectivity*" (JDBC) proporciona los mecanismos necesarios para conectar con servicios de bases de datos requeridos. Esto habilita el acceso a datos desde cualquier componente de la aplicación Web de forma uniforme.

"*Remote Method Invocation*" (RMI) proporciona la capacidad de construir aplicaciones distribuidas usando Java. RMI combina tecnologías de serialización y protocolos de llamadas a métodos remotos para homogeneizar las llamadas de métodos locales y remotos permitiendo definir objetos remotos que pueden ser usados como si fuesen locales.

IBM-RSA puede usar "*WebSphere Application Server*" (WAS) (IBM, 2009) como servidor de aplicaciones de la capa de recursos. Desarrollado por IBM proporciona un entorno de ejecución para desarrollar y desplegar los servicios Java EE anteriormente mencionados. Es necesario comentar que la versión incluida con IBM-RSA por defecto es WAS 5.1, la cual no soporta J2EE impidiendo la utilización de los nuevos componentes de Java EE. La documentación proporcionada para instalar y configurar apropiadamente la versión WAS 6.0 dentro de IBM-RSA que da soporte a los nuevos componentes es escasa y conflictiva.

JBoss (Community, 2009) es otro servidor de aplicaciones de la capa de recursos de código abierto que proporciona un entorno de ejecución para los componentes Java EE, y por lo tanto soportado por IBM-RSA. JBoss es fácil de integrar y configurar dentro de IBM-RSA, sin embargo utilizar un servidor de aplicaciones diferente a WAS limita algunas características de IBM-RSA.

En cuanto a bases de datos se refiere, IBM-RSA tiene la capacidad de administrar bases de datos a través de su perspectiva Data. Puede tomar cualquier fichero con sentencias SQL y ejecutarlo contra cualquier base de datos soportada por la aplicación, así como tomar información del esquema de una base de datos y transformarlo dentro de un diagrama UML.

Desarrollo de aplicaciones Web con IBM-RSA

Como se ha comentado, Java EE es un estándar diseñado para simplificar el desarrollo de aplicaciones empresariales (incluyendo aplicaciones Web) por medio de un conjunto de especificaciones que dan soporte al desarrollo de aplicaciones multicapa, así como también proporciona un conjunto de componentes y tecnologías para cada una de las capas.

IBM-RSA ofrece características para el modelado, desarrollo y despliegue de aplicaciones Web con arquitectura multicapa utilizando las especificaciones de Java EE donde los componentes se ubican en diferentes capas y cada capa tiene un rol diferente en el sistema. IBM-RSA soporta las especificaciones Java EE por medio de la inclusión de diferentes *frameworks* y asistentes para la creación y despliegue de los componentes definidos en Java EE, por lo tanto es imprescindible entender estas tecnologías y sus beneficios y cómo son soportadas por IBM-RSA.

Aunque las características de modelado y edición visual están diseñadas para mejorar la productividad, fortalecer el control arquitectónico y facilitar el diseño y desarrollo de aplicaciones Web, se puede decir que la herramienta IBM-RSA en el desarrollo de aplicaciones Web, lejos de encubrir su complejidad, requiere un cierto nivel de profundidad en el conocimiento de las tecnologías relacionadas con el desarrollo de cada capa de las aplicaciones. La documentación proporcionada por IBM es amplia pero difícil de navegar, imprecisa y a veces contradictoria.

Además, para escenarios donde la plataforma final cumple las especificaciones Java EE, IBM-RSA es la herramienta apropiada para arquitectos, diseñadores y desarrolladores; sin embargo, si la plataforma de ejecución final no es soportada por IBM-RSA, esta solo puede ser usada como herramienta de modelado.

2.1.5 ORACLE – Application Development Framework

Oracle Application Development Framework, Oracle-ADF (Oracle, 2010d), es un framework estratégico de desarrollo de aplicaciones Web de Oracle construido en la cima de la

plataforma Java EE (Network, 2007) con el objetivo de ofrecer máxima productividad a los desarrolladores de aplicaciones. El framework proporciona soluciones de infraestructura en distintas capas para las aplicaciones y una forma fácil de desarrollar sobre ellas. Empezaremos sin embargo, examinando el concepto de framework antes de presentar la arquitectura de Oracle-ADF, con el fin de mostrar los conceptos más importantes para su uso y conocer de manera general cada una de sus capas y como es su interacción en el desarrollo de las aplicaciones.

Cuando nos encontramos con un conjunto de API de bajo nivel, como las ofrecidas por Java EE, los programadores naturalmente empiezan a desarrollar patrones comunes y métodos convenientes para colmar los requisitos que se repiten una y otra vez. Con la experiencia adquirida, la mayoría de los programadores cuentan con una especie de “toolkit” personal de código práctico y técnicas que pueden reusar cuando se enfrenten a un problema similar. A veces éstos problemas son tan comunes en toda la industria que el “toolkit” personal es mejorado y/o generalizado convirtiéndole en patrón de diseño reconocido para la manipulación del escenario.

Desde este punto de vista, los frameworks son implementaciones concretas de tales patrones, ocultando la mayor parte de la solución, dejando al programador la tarea limitada de configurar los parámetros del framework a sus necesidades propias y simplificando al máximo la tarea de codificación, aunque esta nunca es eliminada completamente. El ámbito de un framework puede variar, unos están diseñados para tratar con partes específicas en el desarrollo de una aplicación, por ejemplo los framework O/R, diseñados para corresponder estructuras orientadas a objetos a estructuras relacionales. Otros framework sin embargo no esta confinados a suplir soluciones puntuales, han evolucionado para suplir un rango mucho mayor de funcionalidades hasta proveer la infraestructura para aplicaciones completas, éste es el caso del framework Oracle-ADF, el cual provee directamente infraestructura para algunas tareas como también hereda funcionalidades agregando e integrando frameworks específicos. Estos frameworks amplios con capacidad de integrar a otros framework son llamados más adecuadamente meta-framework (Grant, 2011).

Un meta-framework (Grant, 2011) es un framework de desarrollo de aplicaciones que abarca un amplio rango de funcionalidades, no solo directamente sino por medio de la inclusión de múltiples frameworks de propósito específico. Un meta-framework debe dar además la posibilidad a los desarrolladores de seleccionar y acoplar frameworks específicos sin afectar el resto de la aplicación. La interfaz de usuario por ejemplo no debe ser afectada por el framework

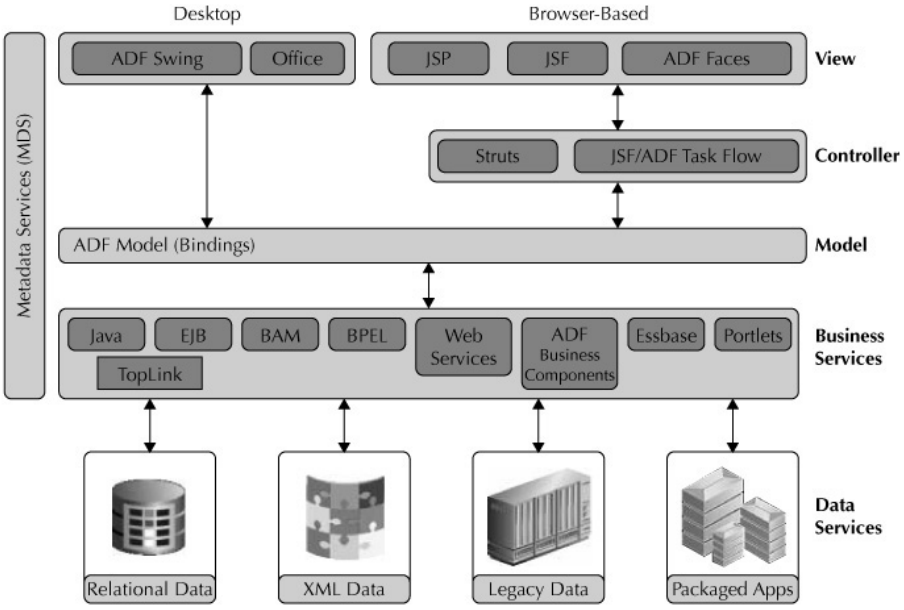
O/R seleccionado. Por lo tanto los meta-framework permiten mucha más flexibilidad en el desarrollo de aplicaciones así como facilidad en el cambio de las tecnologías subyacentes.

Oracle-ADF es un meta-framework que integra un conjunto de frameworks disponibles dentro del universo Java EE con todo lo necesario para que trabajen juntos. Además, si una característica o funcionalidad no está disponible, Oracle-ADF permite agregar paquetes externos y librerías para extender el meta-framework. La figura 2-20 resalta la capacidad de acoplar frameworks de la que dispone Oracle-ADF como meta-framework. Oracle-ADF coordina la selección de frameworks desde un rango de tecnologías que cumplen diversas funciones lógicas según la capa a la que se encuentren orientados, tales como interacción con bases de datos, implementaciones de lógicas de negocio y generación de interfaces de usuario, es decir, permite a los desarrolladores seleccionar la tecnología que prefieran usar cuando implementen cada una de las capas de una aplicación.

La figura 2-20 muestra las variadas opciones disponibles para los desarrolladores cuando construyan aplicaciones con Oracle-ADF. El elemento clave que integra los variados componentes de las aplicaciones Java EE desarrolladas con Oracle-ADF haciendo el desarrollo inmensamente flexible es ADF-Model. La capa ADF-Model actúa como interfaz entre los varios frameworks proveedores de Servicios de Negocio, tales como EJB, Web Services, JavaBeans, TopLink entre otros, y los consumidores de estos servicios, generalmente las interfaces de usuario HTML generadas usando JSP, Java Server Faces (JSF) (Network, 2009a), ADF Faces (Oracle, 2010a), entre otras.

Como comentamos anteriormente, Oracle-ADF está construido en la cima de la plataforma Java EE. Java EE es una plataforma estándar, robusta, escalable y segura que forma la base de muchas de las aplicaciones empresariales actuales (Oracle, 2009). Java EE proporciona un conjunto de especificaciones para construir aplicaciones multicapas usando el lenguaje Java. Existe una relación directa entre la naturaleza robusta de una aplicación y la complejidad de desarrollo requerida para lograrlo, sin embargo, Oracle-ADF simplifica el desarrollo de aplicaciones Java EE minimizando la necesidad de escribir código que implementa la infraestructura de la aplicación permitiendo que los desarrolladores se concentren en los requisitos funcionales de la aplicación a desarrollar. Además las aplicaciones generadas por medio de Oracle-ADF se adhieren a patrones estándar y prácticas recomendadas con un esfuerzo muy reducido (Oracle, 2009).

Figura 2-20. Tecnologías soportadas por Oracle-ADF



Una buena práctica cuando se desarrolla aplicaciones es emplear patrones de diseño. Los patrones de diseño son una forma conveniente de reusar conceptos orientados a objetos entre aplicaciones y entre desarrolladores. Además de los patrones de diseño, los desarrolladores a menudo usan patrones arquitectónicos para construir aplicaciones que funcionen de una forma estándar. Uno de los patrones arquitectónicos más frecuentemente usado es el patrón Model-View-Controller (MVC) (Fowler, 2003). Las aplicaciones desarrolladas con Oracle-ADF cubren todas las capas de un desarrollo basado en el patrón MVC. Una aplicación MVC considera tres roles:

- Modelo: implementa la interacción con las fuentes de datos y ejecuta la lógica del negocio. El Modelo representa el dominio de la aplicación, está compuesto por objetos no visuales que contienen los datos y comportamiento que son usados por componentes de la Vista.
- Vista: contiene los objetos que muestran el estado del Modelo en la interfaz de usuario. El ámbito de la Vista se restringe solo a la forma de visualizar la información del Modelo.

- Controlador: administra el flujo de la aplicación y actúa como interfaz entre la capa Vista y la capa Modelo. Los objetos que contiene el Controlador toman las entradas del usuario, manipulan el Modelo y hacen que la Vista se actualice apropiadamente.

Separar las aplicaciones en estos tres roles simplifica el mantenimiento y la reutilización de componentes. Oracle-ADF además produce una arquitectura orientada a servicios (SOA) (Oracle, 2010c) separando del Modelo los componentes concernientes con la lógica del negocio, alojándolos en una capa llamada capa de Servicios de Negocio. Las capas en las que está basada la arquitectura Oracle-ADF son (Oracle, 2009):

- Capa de Servicios de Negocio, contiene a todos los servicios que proporcionan acceso a datos desde diferentes fuentes y ejecutan la lógica del negocio.
- Capa Modelo, proporciona una abstracción en la cima de la capa de Servicios de Negocio que habilita a las capas Vista y Controlador a trabajar con diferentes implementaciones de Servicios de Negocio en una forma consistente.
- Capa Vista, implementa la interfaz de usuario de la aplicación. Puede ser cliente Web, cliente de escritorio basado en Swing, hoja de cálculo MS Excel, implementación cliente para dispositivos móviles, entre otros.
- Capa Controlador, administra el flujo de la aplicación y maneja las entradas de usuario. Por ejemplo cuando se hace clic sobre un botón “búsqueda” de una página, el controlador determina que acción ejecutar (hacer la búsqueda) y hacia donde navegar (la página de resultados).

Cabe destacar que esta capa orientada a servicios, no tiene por qué implementarse cmo servicios web, pudiendo implementarse con simples servicios de aplicación (Alur et al, 2003). Oracle-ADF además de permitir seleccionar las tecnologías de implementación también permite a los desarrolladores seleccionar el estilo de desarrollo (definición declarativa de propiedades y/o meta datos, visual ó por código), el entorno de desarrollo (Oracle-JDeveloper (Grant, 2011) u otro IDE tal como Eclipse (Eclipse, 2009)) y el ambiente de despliegue (cualquier servidor compatible con Java).

Un aspecto crítico que hace un framework de desarrollo útil es tener una herramienta de desarrollo que simplifique la creación de aplicaciones usando ese framework. Oracle-ADF

proporciona una experiencia de desarrollo visual y de definición declarativa de propiedades a través de la herramienta de desarrollo Oracle-JDeveloper 11g (Grant, 2011). Para crear una aplicación basta con arrastrar y soltar los controles deseados dentro de la página de diseño e indicar que tipo de componente debe representar que datos. Oracle-JDeveloper esta integrado por una serie de herramientas que permiten el desarrollo visual y de definición declarativa de propiedades para cada una de las capas de las aplicaciones desarrolladas en Oracle-ADF. Estas herramientas de desarrollo están sincronizadas dentro del IDE de Oracle-JDeveloper de forma que el editor visual, inspector de propiedades y los modelos están siempre sincronizados con el código fuente. De esta manera los desarrolladores pueden escoger el estilo de desarrollo (arrastrar y soltar, definición declarativa de propiedades o editando el código directamente).

En Oracle-JDeveloper, el desarrollo de la capa de Servicios de Negocio, que puede incluir los componentes proveedores de Servicios de Negocio vistos anteriormente, se puede hacer por medio de asistentes y con un simple clic pueden ser expuestos como interfaces ó Web Services. Además, se puede cumplir este cometido en forma de modelado visual y declarativo.

El desarrollo de la capa Vista y Controlador también esta plenamente soportada por Oracle-JDeveloper, para modelar el flujo de páginas del controlador proporciona un método de modelado visual usando simples componentes arrastrar y soltar dentro del diagrama. Además proporciona un editor visual para interfaces de usuario permitiendo el desarrollo WYSIWYG.

Oracle-JDeveloper también proporciona una forma muy fácil para enlazar los componentes desde la capa de Servicios de Negocio al Controlador y a los componentes de la capa Vista. Para esto proporciona una Paleta de Control de Datos que contiene una vista de los componentes de la capa de Servicios de Negocio. Los desarrolladores pueden simplemente arrastrar y soltar la vista de estos componentes y enlazarlos a los componentes de la interfaz de usuario. El mismo mecanismo se aplica para enlazar acciones del Controlador a métodos definidos en la capa de Servicios de Negocio.

Desarrollo de aplicaciones Web con ORACLE-ADF

Un framework de desarrollo no se encontraría completo sino incluyera una metodología de desarrollo. El método de desarrollo Oracle-ADF (ADF Development Method) (Oracle, 2010b) contempla cuatro pasos:

- Crear espacio de trabajo, consiste en crear una aplicación workspace (*.jws) que aloje los proyectos asociados a nuestra aplicación. Si se elije como Application Template “Web Application Default” se crearan automáticamente los proyectos Model y ViewController. Cada uno de ellos almacenará los componentes asociados a la respectiva capa.
- Crear la capa de Servicios de Negocio y la capa Modelo, durante esta etapa se crearán los componentes que serán mapeados contra las fuentes de datos y generará la capa de persistencia de la que dispondrá nuestra aplicación.
- Crear la capa Vista y la capa Controlador, consiste en desarrollar las páginas de interfaz de usuario que expondrá la aplicación y los flujos entre estas páginas.
- Finalmente se realizan las pruebas y correcciones respectivas.

Un concepto erróneo es pensar que en Oracle-ADF las funcionalidades son implementadas en miles de líneas de código Java generadas automáticamente por los asistentes proporcionados por Oracle-JDeveloper. Este definitivamente no es el caso, gran parte del desarrollo de una aplicación en Oracle-ADF es hecha a través de meta datos. Esto significa que por ejemplo, la definición de que la aplicación debería navegar desde la página “A” a la página “B”, es manejada por meta datos definidos en XML. Se puede pensar que los meta datos definidos en XML son tan difíciles de mantener que el código mismo, sin embargo Oracle-JDeveloper permite la manipulación de meta datos de una forma intuitiva y simplificada. En el caso de la navegación entre páginas, el desarrollo es presentado en un editor visual de flujo de páginas como se muestra en la figura 2-21, obviamente, el hecho de cambiar el flujo de páginas visualmente también cambia el fichero XML, pero esto debería ser algo que el desarrollador no necesita ver.

En la figura 2-21 se muestra como usar el componente ADF Task Flow para definir las reglas de navegación entre dos páginas. En el explorador de proyectos, bajo el proyecto ViewController se encuentra el nodo Page Flows, aquí se definirá la navegación de la aplicación. Se debe crear un nuevo diagrama dentro de este nodo y arrastrar y soltar las páginas para las cuales definiremos las reglas de navegación. Desde la paleta de componentes seleccionar el componente Control Flow Case y unir con líneas dirigidas que indican el flujo de navegación las páginas del diagrama. Cada línea de navegación debe tener un nombre.

Ahora bastará con incluir en las páginas implicadas en el flujo de navegación los controles de interfaz de usuario responsables de ejecutar la navegación. En la figura 2-22 se vincula un botón “Submit” con la acción “goQuery” de la figura 2-21.

Figura 2-21. Definición visual de navegación con ADF Task Flow

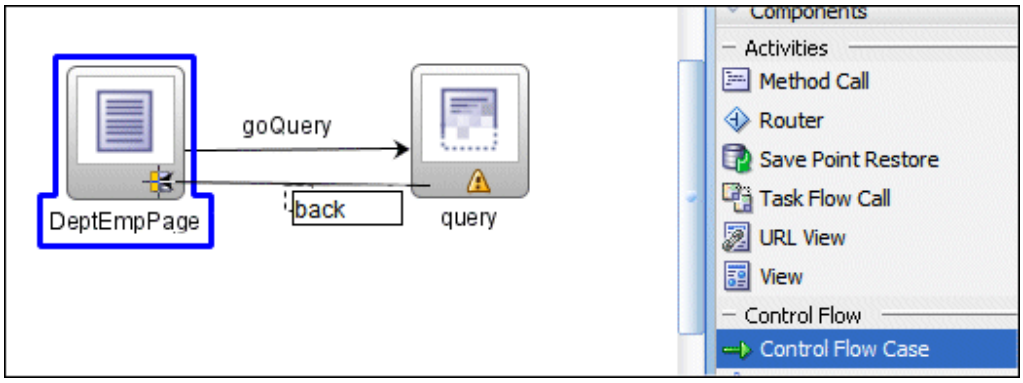
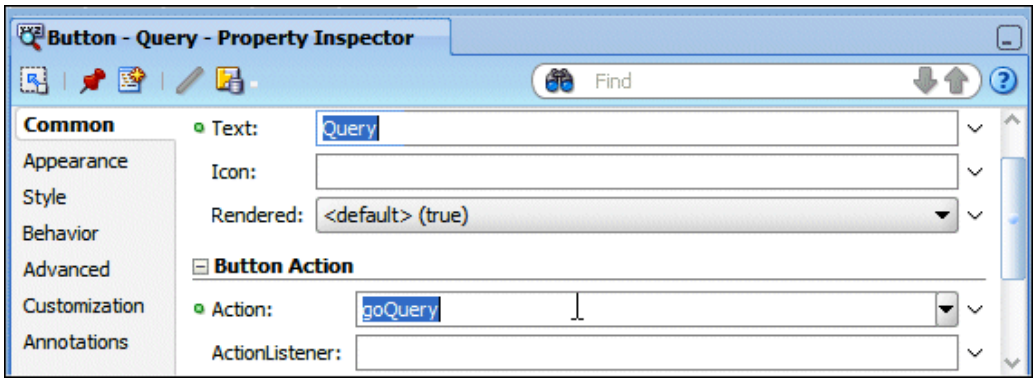


Figura 2-22. Asignación de componentes de UI a acciones de navegación



2.1.6 UML Web Application Extension – UML WAE

Una de las características más relevantes de la notación UML (OMG-UML, 20110) es su capacidad para definir especializaciones del lenguaje. La necesidad de implementar aplicaciones Web a través de complejas arquitecturas con múltiples capas de componentes, ha supuesto un gran reto al abordar su modelado y especificación. En (Conallen, 2003) se presenta una extensión de la notación UML denominada *Web Application Extension – WAE* que permite rentabilizar toda la gramática interna de UML para modelar aplicaciones con elementos específicos de la arquitectura de un entorno Web.

Sus autores hacen énfasis en diferenciar implementación de un *Sitio Web* con desarrollo de una *aplicación Web*. Un *Sitio Web* es relativamente estático, en cambio, una *aplicación Web* es dinámica, dispone de una lógica de negocio que puede reaccionar y alterar su estado a partir de la interacción con los usuarios. Su desafío es la complejidad, ya que requiere implementar una arquitectura que se adapte a los cambios constantes, que facilite su ágil integración con otros sistemas y que resuelva picos variables de interacción con un buen rendimiento. Para afrontar este desafío los autores proponen modelar los elementos específicos de las aplicaciones Web con UML.

Cuando se intenta modelar aplicaciones Web con UML se encuentra que algunas de sus características no encajan dentro de ningún elemento de modelado de UML estándar. Por ejemplo, UML estándar no facilita la tarea de diferenciar entre componentes que se ejecutan en el lado del servidor y componentes que se ejecutan del lado cliente. Con el fin de conseguir una única notación para todo el sistema (componentes Web y componentes tradicionales de las capas internas) UML puede ser extendido usando su propio mecanismo de extensión formal: los *perfiles UML* (Fuentes and Vallecillo, 2004). Los mecanismos que se utilizan para definir los perfiles son: estereotipos (*stereotypes*), restricciones (*constraints*) y valores etiquetados (*tagged values*).

Los estereotipos son adornos que permiten definir un nuevo significado semántico para el elemento de modelado al que se aplica. Los valores etiquetados con pares clave-valor que pueden ser asociados a un elemento de modelado. Las restricciones son reglas para definir modelos bien formados que pueden ser expresadas textualmente o formalmente a través de OCL (OMG-OCL, 2011).

WAE (Conallen, 1999) es un perfil UML que extiende su sintaxis y su semántica para expresar los conceptos específicos del ambiente Web. La extensión WAE ha sido diseñada de tal forma que los componentes específicos del entorno Web pueden ser integrados con el resto del modelo UML del sistema exhibiendo el nivel de abstracción y detalle adecuado para diseñadores, arquitectos y desarrolladores de aplicaciones Web.

El proceso de modelado WAE

El proceso de modelado WAE asume que las aplicaciones Web son aplicaciones centradas en lógica de negocio, por lo tanto los modelos más importantes del sistema deben

hacer énfasis en la lógica y el estado del negocio y no en la capa de presentación. Si las características de la presentación son importantes o de alta complejidad deberían ser modeladas pero independientemente del modelo de negocio.

El lenguaje de modelado UML es el estándar más utilizado para especificar y documentar sistemas intensivos en software, sin embargo dado que las aplicaciones Web cuentan con características especiales como las comentadas anteriormente, es deseable poder contar con un lenguaje más específico para modelar y representar más adecuadamente sus particularidades. Los Perfiles UML constituyen el mecanismo que proporciona el propio UML para expresar más exactamente los conceptos específicos de un determinado dominio de aplicación. Como lo comentamos anteriormente, WAE es un perfil UML creado para caracterizar exactamente las aplicaciones Web.

La lógica de negocio que es ejecutada detrás del servidor Web, es modelada a través de los diagramas de UML estándar (diagrama de casos de uso, diagramas de secuencia, diagramas de componentes, etc.). El perfil WAE es utilizado para los componentes específicos de los entornos Web y la integración con el resto de la aplicación. En concreto, a través del perfil WAE se modela las páginas Web, los enlaces entre páginas Web y el contenido dinámico generado por el servidor y presentado por el cliente Web. De esta forma, en una arquitectura multicapas, UML WAE se utiliza en el modelado de la capa de presentación únicamente.

Para modelar estos elementos de forma consistente con los modelos UML, son mapeados a elementos de UML estándar a los cuales se les aplica adecuadamente los estereotipos, valores etiquetados y restricciones definidos en el perfil WAE. Por ejemplo, las páginas Web son mapeadas a clases UML a las cuales se les aplica los estereotipos `«client page»` si la página está en el lado del cliente, es decir solo contiene código HTML (ó XML) y JavaScript, ó `«server page»` si la página contiene código que será ejecutado por el servidor mientras está siendo preparada. Los enlaces entre páginas representan caminos navegacionales desde una página a otra, por lo tanto son mapeados a asociaciones entre páginas con estereotipos `«Link»`, `«Build»`, `«Submit»`, etc. En las secciones siguientes comentaremos estos estereotipos con más detalle.

El proceso de modelado con WAE hace una clara distinción entre la estructura de navegación por la aplicación y la apariencia de la interfaz de usuario. WAE solo se ocupa de modelar la navegación por la aplicación. Conceptos de la presentación como componentes de

interfaz de usuario no son modelados más allá de la división de la ventana en regiones. Si por necesidades de la aplicación es necesario modelar la interfaz de usuario, este se debe construir como un modelo independiente del modelo construido con WAE.

Modelado de páginas Web con WAE

En el modelado de una aplicación Web es necesario describir todas las páginas Web y sus relaciones. En UML WAE una página Web es cualquier elemento que puede ser servido por un servidor Web. Como lo comentamos anteriormente, en WAE las páginas Web son representadas por clases UML estereotipadas y los enlaces entre ellas como relaciones estereotipadas entre clases. En WAE cada página Web que es construida dinámicamente se modela a través de dos elementos: una clase estereotipada `«server page»` y una clase estereotipada `«client page»`. La clase `«server page»` representa la página conceptual que se ejecuta del lado del servidor cuando un cliente Web hace la petición, y la clase `«client page»` representa la página vista por el cliente Web, en esencia constituye la interfaz de usuario de la aplicación que ha sido creada dinámicamente por la clase `«server page»`.

Como es lógico la clase `«server page»` tiene acceso a los recursos del lado del servidor (componentes en las capas intermedias de la aplicación), sus métodos pueden ser vistos como scripts que se ejecutan del lado del servidor y sus atributos como variables en el ámbito de la página que pueden ser modificados por la lógica que se ejecuta en las capas intermedias de la aplicación, en este sentido, las clases `«server page»` son clases que participan en la lógica de negocio de la aplicación, típicamente toman el rol de controladores que gestionan la actividad de los objetos de negocio para cumplir los objetivos de negocio iniciados a través de una petición de página desde el cliente Web. En la figura 2-23 se muestra un ejemplo de modelado del lado servidor de una aplicación Web con WAE.

La clase `«client page»` tiene un comportamiento y relaciones completamente diferentes. Se relaciona con el navegador Web a través de *Document Object Model* (DOM) (W3C, 2005), con el código JavaScript, controles ActiveX y en general con cualquier plugin que la página especifique. La figura 2-24 muestra un ejemplo de modelado del lado cliente de una aplicación Web con WAE.

Como se dijo anteriormente, las clases `«server page»` y `«client page»` son dos abstracciones de una misma página Web, en WAE se deben relacionar a través de una asociación

dirigida y estereotipada como «build» que significa que la página de servidor construye la página cliente.

Figura 2-23. Modelo del lado servidor de una aplicación Web con WAE

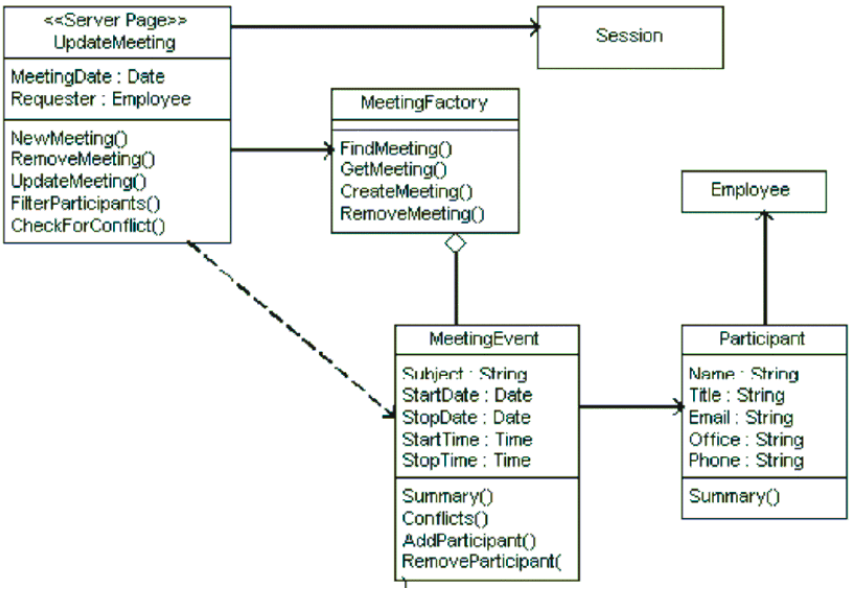
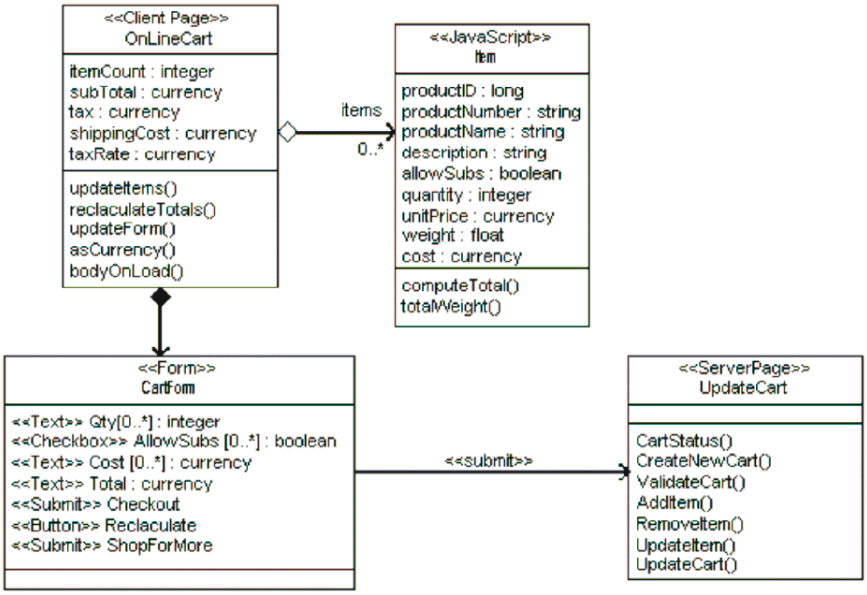
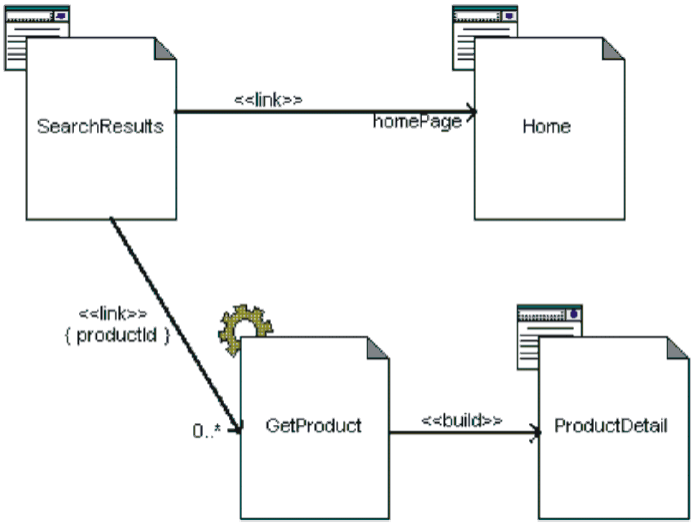


Figura 2-24. Modelo del lado cliente de una aplicación Web con WAE



Otra relación común entre páginas Web es el enlace. Un enlace representa un camino navegacional a través de la aplicación Web. En WAE esta relación se expresa a través de asociación dirigida y estereotipada como «link». Esta asociación siempre debe originarse en una clase «client page» y finalizar en una clase «client page» ó en una clase «server page». Para expresar los parámetros que pueden ser pasados en un enlace se hace uso de valores etiquetados como una lista de pares clave-valor. En la figura 2-25 se muestra un ejemplo de modelado de páginas Web con las relaciones entre ellas en WAE, este modelo también es llamado *mapa navegacional* de la aplicación Web.

Figura 2-25. Modelo de un mapa navegacional de una aplicación Web



El principal mecanismo de entrada de datos en una página Web es el *Form*. En un modelo WAE cada Form se representa por medio de una clase UML estereotipada como «form», debe pertenecer a una clase «client page» y debe especificar la clase «server page» a la cual enviará la información a través de la relación dirigida y estereotipada como «submit». El modelo de la figura 2-24 muestra un ejemplo de cómo modelar un elemento «form».

UML WAE también incluye elementos para caracterizar la interfaz de usuario. Así, clases estereotipadas con la palabra «Frameset» caracterizan ventanas y clases estereotipadas con la palabra «Target» caracterizan regiones de esas ventanas en las que se van a mostrar páginas destino de algún enlace.

Cabe destacar que a pesar de su utilidad, en el seno de una arquitectura multicapa (Alur et al., 2003), UML WAE sólo se utilizaría para modelar elementos de la capa de presentación, sirviendo UML plano para modelar el resto de capas.

2.1.7 Navigation Maps Modeling – NMM

NMM (Navarro et al., 2008b) es la única notación desarrollada explícitamente teniendo en cuenta la arquitectura milticapa (Alur et al., 2003). NMM sirve para modelar la capa de presentación de las aplicaciones Web. En NMM dicha capa está compuesta por la estructura navegacional de la aplicación, la descripción de la interfaz de usuario en términos de *regiones* y la relación entre ambas. La estructura navegacional se representa a través de *mapas navegacionales* que describen todas las secuencias posibles de páginas Web que pueden ser visitadas por los usuarios. La interfaz de usuario se modela a través de *diagramas de regiones* que muestran como las ventanas de la interfaz son divididas. La relación entre los mapas navegacionales y los diagramas de regiones se representa a través de *diagramas de mezcla* los cuales describen el acceso navegacional de los usuarios a las páginas a través de la interfaz de usuario.

NMM permite obtener al mismo tiempo los beneficios de las notaciones de diseño de alto y bajo nivel. NMM como notación de alto nivel caracteriza los elementos principales de las aplicaciones Web omitiendo los detalles arquitectónicos (por ejemplo, la presencia ó no de un controlador), haciendo que los modelos sean sencillos y muestren una visión general de la aplicación. NMM como notación de bajo nivel permite ser fácilmente traducida en componentes concretos de aplicaciones Web. Además, dado que la notación NMM proporciona una semántica formal para los elementos de sus modelos, estos puede ser fácilmente transformarlos a modelos UML-WAE (Conallen, 1999) que caracterizan detallados diseños arquitectónicos. Así, dependiendo de la complejidad de una aplicación Web modelada con NMM se puede elegir la arquitectura más adecuada en el momento de ser transformada a modelos UML-WAE.

Como lo comentamos anteriormente, NMM se centra en el modelado de la capa de presentación de las aplicaciones Web, no se ocupa de modelar los componentes del lado del servidor responsables de construir y dirigir el proceso navegacional, estos artefactos pueden ser automáticamente definidos de acuerdo a la arquitectura seleccionada cuando los modelos NMM sean convertidos a modelos UML-WAE. NMM emplea tres tipos de diagramas: *diagramas de*

página, *diagramas de región* y *diagramas de mezcla*, los cuales cuentan tanto con una definición semántica formal como representación visual. Por simplicidad, a continuación detallaremos cada uno de ellos desde el punto de vista visual.

Diagramas de página

En NMM los diagramas de página definen la estructura navegacional de una aplicación Web a través de mapas navegacionales, los cuales están compuestos por las páginas Web de la aplicación y sus relaciones. En este contexto, una página Web se define como una página que es administrada por un navegador Web, es decir páginas que están desarrolladas con algún lenguaje de marcado como HTML (W3C, 2011b) ó XML (W3C, 2011a). Las relaciones entre las páginas Web se representan por medio de enlaces que van desde *anclas* contenida en paginas Web fuente hasta páginas Web destino. Un ancla representa un dispositivo dentro de una página Web que tiene la capacidad de iniciar una petición http a un servidor Web generando normalmente algún proceso computacional del lado del servidor que produce una página Web destino.

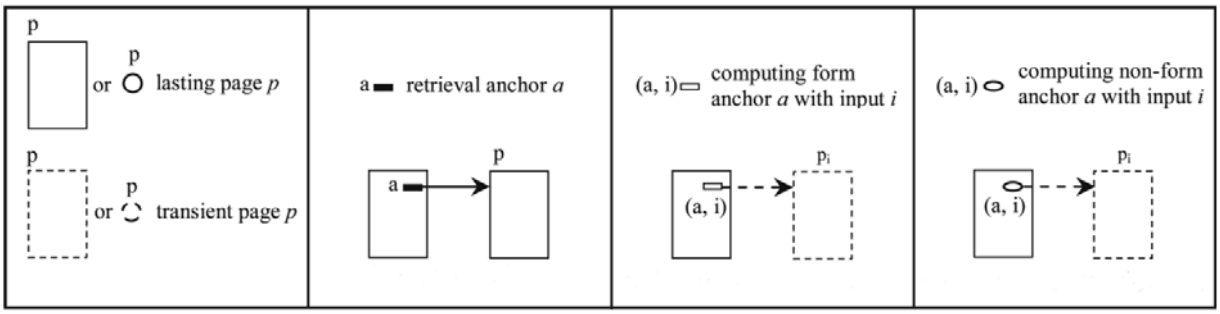
NMM diferencia dos tipos de páginas: páginas estáticas *(lasting pages)* y páginas dinámicas *(transient pages)*. Páginas estáticas son aquellas páginas que están completamente definidas antes de cualquier interacción con la aplicación, es decir no requieren de procesos computacionales para su elaboración. Páginas dinámicas son aquellas que son construidas a través de componentes computacionales invocados por el servidor Web, es decir son generadas por medio de procesos computacionales lanzados como resultado de la interacción con la aplicación Web.

De la misma forma, NMM diferencia dos tipos de anclas: anclas de recuperación (*retrieval anchors*) y anclas computacionales (*computing anchors*). Las anclas de recuperación dan acceso a las páginas estáticas, mientras que las anclas computacionales dan acceso a las páginas dinámicas. Además las anclas computacionales son clasificadas como *form computing anchors* los cuales representan botones de envío dentro de *Web forms* y como *non-form computing anchors* que representan anclas que generan peticiones desde cualquier proceso computacional diferente a los botones de envío. La figura 2-26 muestra la notación gráfica de los componentes de los diagramas de páginas NMM.

Como lo comentamos anteriormente, los diagramas de páginas NMM definen la estructura navegacional de una aplicación por medio de mapas navegacionales que muestran las

relaciones entre páginas en términos de transiciones de una página a otra ocultando los componentes del lado del servidor responsables de construir la página destino. En NMM las anclas computacionales son los componentes que representan estos artefactos del lado del servidor involucrados en el enrutamiento de una página a otra y en la construcción de la página destino. Cuando los modelos NMM evolucionen a modelos con una arquitectura específica, las anclas computacionales serán la base para la definición de los componentes del lado del servidor de estos procesos computacionales.

Figura 2-26. Notación gráfica de los componentes de los diagramas de páginas NMM



Los diagramas de páginas NMM no se ocupan del modelado de la estructura interna de las páginas ni del modelo de datos de las aplicaciones, se asume que las páginas dinámicas generadas por los anclas computacionales incluyen la información extraída desde el modelo de datos de la aplicación, más adelante esta relación entre el modelo de datos y las páginas se puede hacer explicita a través de objetos de transferencia de datos. La figura 2-27 muestra un ejemplo de un diagrama de páginas NMM a través de la notación gráfica definida en la figura 2-26.

Diagramas de región

Los diagramas de región NMM muestran las diferentes regiones en las cuales las ventanas de la interfaz de usuario son divididas. Estos diagramas incluyen la definición de regiones, ventanas y la relación de agregación entre ellas que define cuando una región pertenece a una ventana. Los elementos ventana y región son utilizados porque representan conceptos generales propios de un modelo independiente de la plataforma. La figura 2-28 muestra la definición visual de los elementos de los diagramas de región NMM y la figura 2-29 muestra un ejemplo ilustrativo de modelado con diagramas de región NMM.

Figura 2-27. Diagrama de páginas NMM

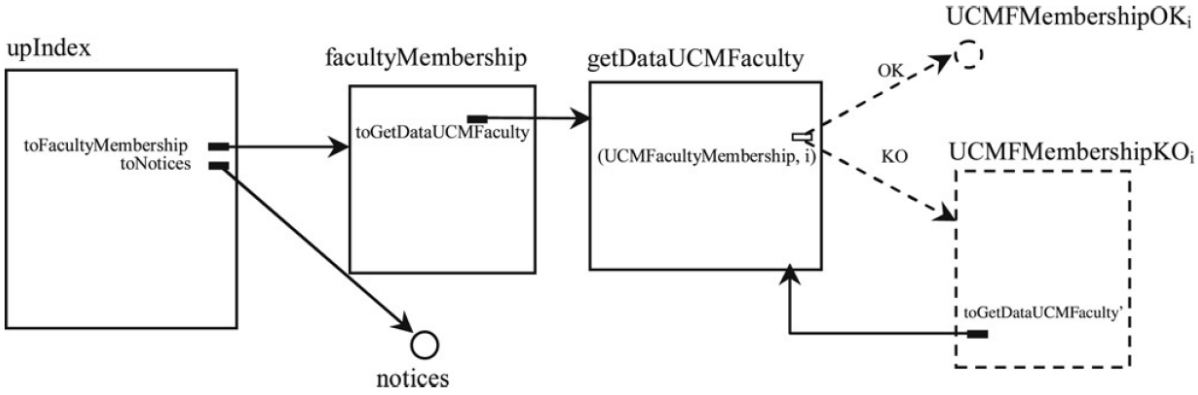


Figura 2-28. Notación gráfica de los componentes de los diagramas de región NMM

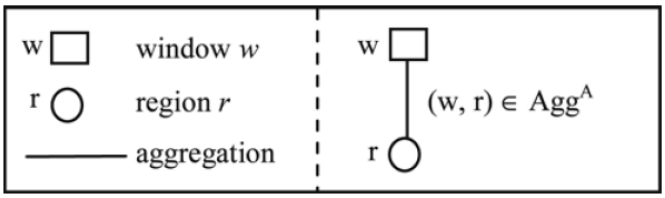


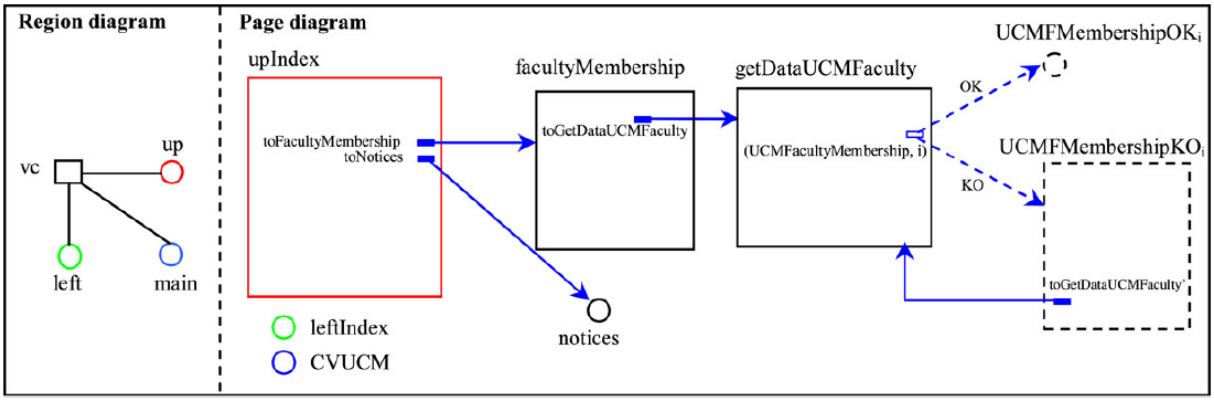
Figura 2-29. Diagrama de región NMM

Diagramas de mezcla

Los diagramas de mezcla NMM describen el acceso navegacional a las páginas de la aplicación a través de la interfaz de usuario. Para esto, el diagrama de mezcla relaciona los diagramas de página con los diagramas de región proporcionando la mayor flexibilidad. Un mismo diagrama de página puede ser mapeado o adaptado a diferentes diagramas de regiones; de la misma forma, un diagrama de región puede ser usado con diferentes diagramas de páginas.

En la figura 2-30 se muestra un diagrama de mezcla NMM que relaciona el diagrama de páginas de la figura 2-27 y el diagrama de regiones NMM de la figura 2-29 usando colores. De acuerdo a este diagrama, el color de la página "*upIndex*" indica que es la página por defecto de la región "*up*". Los colores también relacionan las anclas (definidas en el diagrama de páginas) con las regiones, si el usuario navega a través de los enlaces que unen las páginas "*upIndex*" con las páginas "*facultyMembership*" ó "*notices*", estas páginas aparecerán en la región "*main*". El diagrama muestra que la región "*main*" es la región destino de todas las páginas que son alcanzadas a través de cualquier enlace.

Figura 2-30. Diagrama de mezcla NMM



En el trabajo recogido en esta memoria, la notación NMM pasa de ser una notación formal con una representación gráfica, a ser una extensión UML. La potencia de modelado en ambos casos es prácticamente equivalente, pudiendo pasar así de modelos formales a representaciones UML y viceversa de manera automática. Sin embargo, para diferenciar el uso de NMM con carácter formal, o como extensión UML, este trabajo introduce la notación NMMp (*NMM profile*), como la representación de NMM a través de un perfil UML.

2.2 Análisis de las diferentes aproximaciones de modelado para aplicaciones Web

En esta sección se compara NMMp con diferentes notaciones de diseño. En la comparación incluida en esta sección se han introducido notaciones web adicionales a las analizadas en este capítulo. No se ha proporcionado una descripción detalladada de las mismas por no sobrecargar el estado del arte, pero se ha estimado oportuno incluirlas en esta sección para enriquecer la comparación de NMMp con diversas aproximaciones de modelado Web. Sin embargo, esta sección incluye partados específicos de comparación entre NMMp y las aproximaciones analizadas en este capítulo.

En este comparativa, las notaciones se agrupan en los siguientes cuatro niveles de abstracción:

- Nivel IV: notaciones basadas en *Modelos Específicos de la Plataforma (PSM)*. En esta categoría, los diagramas de diseño son modelos específicos de la plataforma y además, no son derivados desde *Modelos Independientes de la Plataforma (PIMs)*. Esta categoría incluye notaciones usadas por herramientas como: *Oracle ADF* (Oracle, 2011) y los diagramas Web generados con *IBM Rational Software Architect* (IBM, 2010). Nótese que IBM Rational Software Architect es una herramienta UML CASE de propósito general. Sin embargo, incluye notaciones de diseño específicas para la caracterización de las aplicaciones Web.
- Nivel III: notaciones basadas en *Modelos Independientes de la Plataforma (PIMs)* pero dependientes de la arquitectura. En esta categoría los PIM son definidos, pero incluyen detalles arquitectónicos específicos como Model 1 ó Model 2. Esta categoría incluye notaciones como UML estándar (Arlow and Neustadt, 2005) y UML-WAE (Conallen, 1999).
- Nivel II: notaciones basadas en *Modelos Independientes de la Plataforma (PIMs)*. En esta categoría los PIM son independientes de los detalles arquitectónicos, pero son transformados en PIMs dependientes de la arquitectura y en modelos PSM. Esta categoría incluye notaciones como *NMMp*, *OOH4RIA* (Melia et al., 2010) y *UWA* (Distante et al., 2007).

- Nivel I: notaciones basadas en únicamente *Modelos Independientes de la Plataforma (PIMs).* En esta categoría los modelos PIM son directamente transformados a código. De esta forma no son necesarios otros modelos PIM o PSM. Esta categoría incluye notaciones como: *RUX* (Linaje et al., 2007), *UWE* (Koch and Kraus, 2003) y *WebML* (Bozzon et al., 2006).

En la tabla 2-1, estas notaciones son analizadas tomando en cuenta las siguientes diez características:

- *Presencia de PIMs independientes de la arquitectura*. Esta característica es muy valiosa porque proporciona una vista abstracta de la aplicación facilitando la validación a nivel de usuario (Bozzon et al., 2006; Koch and Kraus, 2003; Linaje et al., 2007; Melia et al., 2010; Navarro et al., 2008a).
- *Presencia de PIMs y/o PSMs adicionales.* Esta característica es importante porque proporciona semántica adicional a los abstractos PIM, mejorando la mantenibilidad del código por parte de los programadores (Distante et al., 2007; Melia et al., 2010; Navarro et al., 2008a).
- *Descripción de otras capas en adición a la capa de presentación.* Una aplicación Web multicapas está compuesta por más de una capa. Cuantas más capas son descritas por una notación, más detalles de la aplicación es revelado, por lo tanto debe ser considerada más valiosa la notación (Bozzon et al., 2006; Distante et al., 2007; Koch and Kraus, 2003; Melia et al., 2010).
- *Compatibilidad explicita con el uso de patrones arquitectónicos y patrones SOA.* El uso de los patrones mejora el entendimiento y mantenibilidad de las aplicaciones (Alur et al., 2003; Erl, 2009b; Fowler, 2003; Monday, 2003). Por lo tanto, su presencia es de vital importancia en las aplicaciones empresariales que tienen que ser mantenidas en el tiempo.
- *Descripción de interfaces de usuario RIA.* Las interfaces de usuario RIA permiten mejorar la interacción de los usuarios con las aplicaciones Web. Por lo tanto están incrementando su popularidad entre los desarrolladores y diseñadores (Bozzon et al., 2006; Koch and Kraus, 2003; Linaje et al., 2007; Melia et al., 2010).

- *Descripción de flujos navegacionales.* Los flujos navegacionales ayudan a los usuarios, diseñadores y programadores a usar, diseñar y construir aplicaciones Web más fáciles de usar, por lo tanto se han convertido en un componente importante de las notaciones de diseño (Bozzon et al., 2006; Koch and Kraus, 2003; Linaje et al., 2007; Melia et al., 2010; Navarro et al., 2008a).
- *Ocultamiento de artefactos computacionales.* La presencia explicita de artefactos computacionales (por ejemplo, *comandos* de un *controlado de aplicación* (Alur et al., 2003)) dificultan el entendimiento de los flujos navegacionales para los usuarios y diseñadores (Navarro et al., 2008a). Por lo tanto, su exclusión mejora el entendimiento de las notaciones Web.
- *Generación de aplicaciones "listas" para ser ejecutadas.* Algunas notaciones incluyen herramientas que permiten la generación de aplicaciones *listas* para ser ejecutadas mezclando diagramas de diseño con componentes de código. Esto da el valor añadido a los diagramas de diseño (Bozzon et al., 2006; IBM, 2010; Koch and Kraus, 2003; Melia et al., 2010; Oracle, 2011) pero acopla el mantenimiento del código a una herramienta específica.
- *Soporte para herramientas CASE UML de propósito general.* Cada notación puede construir su propia herramienta CASE. Sin embargo, el uso de herramientas CASE de propósito general facilita el uso de la notación y desacopla el mantenimiento de la notación de una herramienta concreta. (Koch and Kraus, 2003; Navarro et al., 2008a).
- *Mantenimiento de código independiente de entornos de desarrollo específico.* Acoplar el mantenimiento del código a un entorno de desarrollo concreto es un problema (Pressman, 2009). El mantenimiento del código construido sin diagramas de diseño es una labor muy dificultosa (Booch et al., 2007; Pressman, 2009; Sommerville, 2010). Sin embargo, la presencia de diagramas de diseño abstracto orientado a la aplicación más que orientado al código puede dificultar los procesos de diseño y mantenimiento. Por lo tanto la dependencia de una herramienta concreta que proporcione una transformación directa desde modelos PIMs abstractos a código debería ser evitada. Lo más deseable sería poder mantener el código sin depender de herramientas concretas.

Tabla 2-1. Comparación de NMMp con otros enfoques.

Nivel de Abstracción	Enfoque	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	(viii)	(ix)	(x)
I. Solo basadas en PIMs	RUX	✓	×	×	×	✓	×	✓	×	×	×
	UWE	✓	×	✓	×	✓	✓	×	✓	✓/×	×
	WebML	✓	×	✓	×	✓	✓	✓/×	✓	✓/×	×
II. Basadas en PIMs	NMMp	✓	✓	✓/×	✓	×	✓	✓	×	✓	✓
	OOH4RIA	✓	✓	✓	✓/×	✓	✓	×	✓	×	×
	UWA	✓	✓	✓	✓/×	×	✓	×	✓	×	×
III. Basadas en PIMs dependientes de Arq.	UML WAE	×	✓	✓/×	✓	×	✓	×	×	✓	✓
	UML	×	✓	✓	✓	✓	✓/×	×	×	✓	✓
IV. Basadas en PSMs	IBM RSA	×	×	✓	✓	✓/×	✓	×	✓	×	✓/×
	Oracle ADF	×	×	✓	✓/×	✓	✓	×	✓	×	×

(i). Presencia de PIMs independientes de la arquitectura.
(ii). Presencia de PIMS y/o PSMs adicionales.
(iii). Descripción de otras capas en adición a la capa de presentación.
(iv). Compatibilidad con el uso explicito de patrones arquitectónicos multicapas y SOA.
(v). Descripción de interfaces de usuario RIA.
(vi). Descripción de flujos navegacionales.
(vii). Ocultamiento de artefactos computacionales.
(viii). Generación de aplicaciones *listas* para se ejecutadas.
(ix). Soporte a herramientas CASE UML de propósito general.
(x). Mantenimiento de código independiente de entornos de desarrollo específicos.

Los enfoques de Nivel I son notaciones abstractas que, excepto RUX, cubren todos los aspectos de las aplicaciones Web. Sin embargo, la abstracción de sus diagramas de diseño dificulta la implementación directa de estos diagramas. De esta forma, estos enfoques incluyen entornos de desarrollo específico que permiten la generación de código, acoplando su mantenimiento a la herramienta de generación. Sin embargo, UWE y WebML pueden ser usados con herramientas CASE UML de propósito general, pero limitando su capacidad de generación de aplicaciones. Además los patrones arquitectónicos multicapas y SOA no están presentes en estos enfoques, por ejemplo, en WebML el movimiento de datos entre capas se da usando *elementos XML* (W3C, 2011a) en lugar de Objetos de Transferencia de Datos. Los enfoques de Nivel I son el mejor ejemplo de enfoques basados en notación para el desarrollo de aplicaciones Web.

Los enfoques de Nivel II intentan transformar los abstractos PIMs a modelos PIM más específicos o a PSMs. En OOH4RIA la transformación a PSM afecta solo a los diagramas de la capa de presentación, quedando los diagramas navegacionales aún demasiado abstractos y siendo directamente transformados a código. De forma que los inconvenientes en el mantenimiento del Nivel I también están presentes. En UWA, cada diagrama abstracto PIM es transformado a PSM, pero como en el caso de OOH4RIA, el uso de patrones arquitectónicos multicapas y SOA están

restringidos a los existentes en el framework. En ambos casos, el uso de patrones arquitectónicos multicapas y SOA no es explícitamente estimulado.

Los modelos PIMs NMMp solo caracterizan elementos de la capa de presentación y son transformados en diagramas PIMs UML-WAE. Estos diagramas pueden ser integrados con diagramas UML estándar que describan el diseño del resto de las capas de la aplicación Web. De esta forma, cualquier patrón de capa de negocio, de integración o de recursos puede ser descrito en UML e integrado con los diagramas UML-WAE generados desde la notación NMMp. Además, NMMp puede ser usado con una herramienta CASE UML de propósito general que soporte transformaciones QVT, tal como Borland Together. Así, NMMp intenta encontrar un equilibrio entre el desarrollo basado en notación y el desarrollo basado en patrones, por lo cual puede ser definido como un enfoque mixto.

La principal limitación de NMMp es que no incluye soporte para interfaces RIA. Esta característica no ha sido en principio incluida porque fuerza la presencia de características especificas de los clientes Web, provocando posibles problemas de compatibilidad entre navegadores, como en el caso de otras tecnologías propias de los clientes Web (Alur et al., 2003; Fowler, 2003). Sin embargo, en el presente, se planea la inclusión de características RIA en NMMp. Finalmente, en NMMp la generación de código esta restringida a la proporcionada por la herramienta CASE UML de propósito general, o a las transformaciones proporcionadas por el usuario, por lo cual usualmente no se generan aplicaciones *listas* para ser ejecutadas.

Los enfoques de Nivel III proporcionan PIMs dependientes de la arquitectura. Los diagrmas UML-WAE carecen de abstracción (Bozzon et al., 2006), convirtiéndose en diagramas muy complejos para aplicaciones con arquitectura Model 2 por ejemplo (Navarro et al., 2008a). Sin embargo, los diagramas UML-WAE pueden ser usados en combinación con los diagramas NMMp para incrementar su nivel de abstracción y mostrar al mismo tiempo detalles arquitectónicos necesarios para los desarrolladores.

Finalmente, los enfoques de Nivel IV son dependientes de tecnologías específicas. IBM-RSA proporciona diagramas de diseño para construir aplicaciones J2EE y Struts. Oracle –ADF aporta un entorno visual completo para la construcción de aplicaciones J2EE usando casi todas las tecnologías existentes para esta plataforma. Sin embargo, los diagramas de IBM-RSA y Oracle-ADF son específicos de la plataforma y muy ligados a la tecnología J2EE (notar que, como se mencionó anteriormente, IBM-RSA es una herramienta CASE UML de propósito

general, sin embargo, incluye notaciones de diseño específicos para la caracterización de aplicaciones Web). En este sentido, están más cerca de los entornos de programación visual que de las herramientas CASE. Además, la inclusión de patrones multicapas y SOA está restringido a los soportados por estas herramientas.

A continuación, se compara con mayor profundidad la notación NMM/NMMp y el resto de notaciones analizadas en este capítulo.

2.2.1 OOH y el enfoque NMMp

Como hemos visto, OOH es un método de modelado genérico para el desarrollo de interfaces Web que extiende los modelos generados previamente con el método OOM. En particular, OOH se basa en el diagrama de clases UML de OOM para generar los modelos navegacional y de presentación. A diferencia del enfoque NMMp que se centra en el modelado de la estructura navegacional y la descripción de la interfaz de usuario independientemente del modelado conceptual de la aplicación.

OOH basa el modelo navegacional en los requisitos de usuario sin tener en cuenta en principio si la información es presentada en una simple página Web ó dividida en varias, esta decisión es pospuesta hasta la etapa de refinamiento y tomada por medio de la aplicación de patrones de interfaz. NMMp sin embargo, describe el modelo navegacional en términos de páginas Web que serán expuestas al usuario sin mencionar su contenido navegacional, esta es una característica importante que ayuda a los desarrolladores a tener una visión global de la aplicación durante el proceso de desarrollo y sirve como documentación a los usuarios de la aplicación.

En OOH el modelo de presentación se ocupa de modelar la apariencia gráfica de las páginas y de los elementos contenidos en ellas por medio de plantillas, NMMp da la flexibilidad de construir libremente las páginas de presentación centrándose solo en definir las regiones de la interfaz de usuario y su relación con los mapas navegacionales.

OOH hace posible la generación de la arquitectura Web de forma automática e independientemente de la tecnología de implementación, pero con la desventaja de que los detalles arquitectónicos no se hacen explícitos, el código generado puede ser visto como una interfaz Web que se conecta a los módulos lógicos de una aplicación desarrollada a través del método OOM. Los modelos NMMp pueden ser vistos como modelos de alto nivel porque son

independientes de la arquitectura, sin embargo tienen la ventaja de que pueden ser fácilmente transformados en modelos UML-WAE que permiten caracterizar detalladamente el diseño arquitectónico. Además, una vez transformados los modelos NMMp en modelos UML-WAE cuentan con todas las ventajas del modelado UML.

2.2.2 El proceso de desarrollo UWE y el enfoque NMMp

Como hemos visto, UWE propone una notación de dominio específico para la representación gráfica de aplicaciones Web y un método MDD que cubre todo el desarrollo de las aplicaciones Web, desde la especificación de requisitos hasta la generación de código. A diferencia del enfoque NMMp que se centra en el modelado de la capa de presentación de las aplicaciones Web que incluye la estructura navegacional, la descripción de la interfaz de usuario y la relación entre las dos, UWE considera el modelado navegacional solo como un paso más en el desarrollo de las aplicaciones Web.

En UWE el modelo navegacional esta directamente relacionado con el modelo conceptual creando una dependencia entre las entidades y/o procesos de negocio y la navegabilidad, en NMMp sin embargo, el modelado navegacional se describe en términos de páginas Web que serán expuestas al usuario sin mencionar en principio como serán construidas, esta es una característica importante que ayuda a los desarrolladores a tener una visión global de la aplicación durante el proceso de desarrollo y sirve como documentación a los usuarios de la aplicación.

Los modelos UWE pueden ser considerados de alto nivel porque describen los elementos principales de una aplicación Web ocultando los detalles arquitectónicos; los modelos NMMp pueden ser vistos como modelos de alto nivel porque también son independientes de la arquitectura, sin embargo tienen la ventaja de que pueden ser fácilmente transformados en modelos UML-WAE que permiten caracterizar detalladamente el diseño arquitectónico.

En UWE el modelo de la presentación esta basado en las entidades del modelo conceptual y del modelo navegacional, proporcionando además descripciones de los componentes de la interfaz de usuario pero sin definir aspectos concretos tales como colores y fuentes de los elementos. Es decir, el modelo de presentación UWE proporciona una vista abstracta de la interfaz de usuario; esto puede ser visto como una limitación si deseamos modelar, por ejemplo, aplicaciones Web enriquecidas (RIA). NMMp da la flexibilidad de

construir libremente las páginas de presentación centrándose solo en definir las regiones de la interfaz de usuario y su relación con los mapas navegacionales.

2.2.3 WebML y el enfoque NMMp

Como hemos visto, WebML es un lenguaje de modelado para diseñar aplicaciones Web, plantea su metodología que cubre todo el proceso de desarrollo, y cuenta con su propia herramienta CASE llamada WebRatio que habilita el desarrollo MDD. A diferencia del enfoque NMMp que se centra en el modelado de la capa de presentación de las aplicaciones Web que incluye la estructura navegacional, la descripción de la interfaz de usuario y la relación entre las dos, WebML considera el modelado navegacional solo como un paso más en el desarrollo de las aplicaciones Web.

En WebML el modelo navegacional puede depender del modelo de procesos de negocio, en NMM sin embargo, el modelo navegacional siempre es independiente tanto de la estructura de datos subyacente como de los procesos de negocio de la aplicación Web.

Tanto WebML como NMMp describen el modelo navegacional en términos de páginas Web que serán expuestas al usuario, sin embargo en WebML el concepto de página no puede existir aisladamente del modelo estructural, es decir el modelo navegacional debe detallar tanto las páginas como su contenido. NMMp no menciona el contenido de las páginas, su objetivo es solo mostrar el diseño navegacional, esta es una característica importante que ayuda a los desarrolladores a tener una visión global de la aplicación durante el proceso de desarrollo y sirve como documentación a los usuarios de la aplicación.

En WebML el modelo de presentación se ocupa de modelar la apariencia gráfica de las páginas y de los elementos contenidos en ellas, NMMp da la flexibilidad de construir libremente las páginas de presentación centrándose solo en definir las regiones de la interfaz de usuario y su relación con los mapas navegacionales.

Los modelos WebML pueden ser considerados de alto nivel porque describen los elementos principales de una aplicación Web ocultando los detalles arquitectónicos; los modelos NMMp pueden ser vistos como modelos de alto nivel porque también son independientes de la arquitectura, sin embargo tienen la ventaja de que pueden ser fácilmente transformados en modelos UML-WAE que permiten caracterizar detalladamente el diseño arquitectónico. Además, una vez transformados los modelos NMMp en modelos UML-WAE cuentan con todas

las ventajas del modelado UML. Así, WebML no proporciona un modelo dependiente de la del código generado, ligando el mantenimiento del código a la herramienta propietaria WebRatio, mientras que NMMp sí proporciona dicho modelo, facilitando el mantenimiento del código con independencia de la propia notación.

2.2.4 IBM-RSA, ORACLE ADF y el enfoque NMMp

Como hemos visto, IBM-RSA es una herramienta CASE que da soporte a las especificaciones Java EE para el desarrollo de aplicaciones Web con arquitecturas multicapas, cubriendo las fases de modelado, desarrollo y despliegue de las aplicaciones, sin embargo, como conclusión del estudio de la herramienta IBM-RSA en el desarrollo de aplicaciones Web, podemos decir que lejos de encubrir la complejidad del desarrollo de las aplicaciones Web, la herramienta requiere un cierto nivel de profundidad en el conocimiento de las tecnologías relacionadas con el desarrollo de cada capa de las aplicaciones.

Las aplicaciones Web modeladas ó desarrolladas con IBM-RSA quedan ligadas a la herramienta no permitiendo que sean mantenidas o evolucionadas con otras herramientas, Además para escenarios donde la plataforma final cumple las especificaciones Java EE, IBM-RSA puede ser una herramienta apropiada para el diseño, desarrollo y despliegue de las aplicaciones, sin embargo, si la plataforma de ejecución final no es soportada por IBM-RSA, esta solo puede ser usada como una herramienta de modelado.

Oracle-ADF es un meta-framework de desarrollo de aplicaciones que integra un conjunto de frameworks disponibles dentro del universo Java EE con todo lo necesario para que trabajen juntos, proporcionando soluciones de infraestructura en distintas capas para las aplicaciones y una forma fácil de desarrollar sobre ellas, aunque es necesario un amplio conocimiento de las tecnologías relacionadas con el desarrollo de cada capa de la especificación Java EE. Las aplicaciones desarrolladas con Oracle-ADF cubren todas las capas de un desarrollo basado en el patrón MVC.

El enfoque NMMp se centra en el modelado de la capa de presentación de las aplicaciones Web independientemente de cualquier herramienta, las aplicaciones modeladas con NMM no hacen mención en principio de ninguna arquitectura ni tecnología de implementación, si bien en cierto que la definición formal de los componentes de NMM facilitan que sean

traducidos a otros modelos que hagan explícitos los detalles arquitectónicos y que la notación NMM ha sido desarrollada explícitamente teniendo en cuenta la arquitectura multicapas.

2.2.5 El proceso de desarrollo con WAE y el enfoque NMMp

El proceso de desarrollo de aplicaciones Web con WAE permite la especificación detallada del diseño arquitectónico, las aplicaciones modeladas con WAE están supeditadas a una arquitectura determinada, además el hecho de incluir detalles arquitectónicos en los modelos hace que estos sean complejos ocultando la visión general de la aplicación. Los modelos NMMp omiten detalles arquitectónicos, son por lo tanto independientes de la arquitectura, en los siguiente etapas de desarrollo dependiendo de la complejidad de las aplicaciones se puede elegir la arquitectura más adecuada.

Los modelos NMMp pueden ser vistos como versiones de alto nivel de los modelos WAE. Dado que la notación NMMp proporciona una semántica práctica para los elementos de sus modelos, estos puede ser fácilmente transformarlos a modelos WAE, es decir modelos NMMp que ocultan detalles arquitectónicos pueden ser transformados automáticamente a modelos WAE describiendo una arquitectura específica.

En WAE los elementos específicos del entorno Web como "*página Web*" ó "*form*" son modelados junto a elementos que representan objetos del dominio de la aplicación modelada. Los modelos NMMp en cambio, modelan la capa de presentación de forma totalmente independiente del resto de las capas de la aplicación Web, esta es una característica clave en las arquitecturas multicapas donde debe existir una clara separación entre la capa de presentación y la capa de negocio.

2.3 Arquitectura Multicapa

Con la introducción del paradigma de la computación distribuida, un gran número de Sistemas de Información han sido construidos para una gran variedad de necesidades de negocio. Una pregunta fundamental es cómo una aplicación distribuida debería desarrollarse desde un punto de vista arquitectónico. El modelo tradicional cliente/servidor contiene dos capas: la capa cliente y la capa servidor. En las implementaciones tradicionales, la capa servidor es un motor de bases de datos, mientras que la capa cliente administra la lógica de negocio y la entrada/salida de

datos típicamente a través de una interfaz de usuario gráfica (Shan and Hua, 2006). La mayoría de los sistemas diseños en este modelo son dirigidos por los datos (*data-driven systems*).

La introducción de los modelos Web ha hecho evolucionar los modelos cliente/servidor en otro paradigma. En este nuevo modelo, el lado del cliente se ha convertido en una capa *delgada* y estandarizada usando lenguajes comunes como HTML (W3C, 2011b), JavaScript y hojas de estilos CSS (W3C, 2008). Casi toda la lógica de negocio es desplegada y procesada en el lado del servidor. La estructura del lado del servidor es usualmente descompuesta en múltiples capas como lo detallaremos a continuación, antes es importante distinguir los conceptos de "capas" (*layers*) y "niveles" (*tiers*), pues es bastante común que se confundan y/o denominen de forma incorrecta (Llorente et al., 2011).

Las capas (*layers*) se ocupan de la división lógica de componentes y funcionalidad sin tener en cuenta la localización física. Por el contrario los niveles (*tiers*) se ocupan de la distribución física de los componentes y funcionalidad en servidores separados, teniendo en cuenta topología de redes y localizaciones remotas. Aunque tanto las capas (*layers*) como los niveles (*tiers*) usan conjuntos similares de nombres (presentación, servicios, negocio y datos), es importante no confundirlos y recordar que solo los niveles (*tiers*) implican una separación física. Se suele utilizar el término "*Tier*" para referirse a patrones de distribución física como "*2-Tier*", "*3-Tier*" y "*N-Tier*".

Cabe destacar que en otros contextos (Alur et al., 2003) el concepto de capa o "tier" se utiliza también para describir agrupación lógica de clases y funcionalidades. En estos contextos la distinción entre agrupación lógica de clases y agrupación de hardware se logra distinguiendo entre *capa lógica* (layer) y *capa física* (tier).

Se ha comprobado que las aplicaciones empresariales de gran escala han sido satisfactoriamente construidas en base a múltiples capas, pues este enfoque ha probado ser escalable, flexible y modular (Hartwich and Berlin, 2001), pero no todas las aplicaciones tienen por qué implementarse en modo N-*Tier*, puesto que hay aplicaciones que no requieren de una separación física de sus niveles. Las capas son agrupaciones horizontales lógicas de componentes de software que forman la aplicación o servicio. Nos ayudan a diferenciar entre los diferentes tipos de tareas a ser realizadas por los componentes, ofreciendo un diseño que maximiza la reutilización y especialmente la mantenibilidad. En definitiva, se trata de aplicar el

principio de “Separación de Responsabilidades” (*SoC - Separation of Concerns principle*) (Fowler, 2003) dentro de una Arquitectura.

El dividir una aplicación en capas separadas que desempeñan diferentes roles y funcionalidades ayuda a mejorar el mantenimiento del código, admite también diferentes tipos de despliegue permitiendo que diferentes capas sean situadas de forma flexible en diferentes servidores o clientes, y, sobre todo, nos proporciona una clara delimitación y situación de dónde debe estar cada tipo de componente funcional e incluso cada tipo de tecnología (Fowler, 2003).

Se ha decidido incluir esta sección en la memoria para contextualizar la aportación de la notación NMMp en el seno de la arquitectura multicapa.

2.3.1 Diseño básico de capas

Los componentes de cada capa deben ser cohesivos y tener aproximadamente el mismo nivel de abstracción. Cada capa de primer nivel debe de estar débilmente acoplada con el resto de capas de primer nivel. La clave de una aplicación en N-Capas está en la gestión de dependencias. En una arquitectura N-Capas tradicional, los componentes de una capa pueden interactuar solo con componentes de la misma capa o bien con otros componentes de capas inferiores. Esto ayuda a reducir las dependencias entre componentes de diferentes niveles. Normalmente hay dos aproximaciones al diseño en capas: *Estricto y Laxo*.

Un *diseño en Capas estricto* limita a los componentes de una capa a comunicarse solo con los componentes de su misma capa o con la capa inmediatamente inferior. La capa J solo podría interactuar con los componentes de la capa J-1, la capa J-1 solo con los componentes de la capa J-2, y así sucesivamente. Un *diseño en Capas laxo* permite que los componentes de una capa interactúen con cualquier otra capa de nivel inferior. La capa J podría interactuar con la capa J-1, J-2… J-(J-1). El uso de la aproximación laxa puede mejorar el rendimiento porque el sistema no tiene que realizar redundancia de llamadas de unas capas a otras. Por el contrario, el uso de la aproximación laxa no proporciona el mismo nivel de aislamiento entre las diferentes capas y hace más difícil el sustituir una capa de más bajo nivel sin afectar a muchas más capas de nivel superior (y no solo a una).

2.3.2 Modelos N-Capas

En (Alur et al., 2003) se propone un diseño de arquitecturas N-Capas, donde, desde el punto de vista más alto y abstracto, un sistema N-Capas es considerado como un conjunto de

servicios relacionados y agrupados en diversas capas como se muestra en la figura 2-31. En las arquitecturas N-Capas es crucial la clara delimitación y separación de las capas, se debe desarrollar un diseño dentro de cada capa que sea cohesivo, delimitando claramente las diferentes capas, aplicando patrones estándar de arquitectura para que las dependencias entre capas se basen en abstracciones y no referenciando una capa directamente desde otra.

Las capas aisladas son mucho menos costosas de mantener porque tienden a evolucionar a diferentes ritmos y responder a diferentes necesidades. Por ejemplo, las capas de datos/recursos evolucionarán cuando evolucionen las tecnologías sobre las que están basadas. Por el contrario, la capa del negocio evolucionará solo cuando se quieran realizar cambios en la lógica de negocio del dominio concreto.

Figura 2-31. Vista de arquitectura simplificada de un sistema N-Capas

A continuación, se describe brevemente cada una de las capas presentes en cualquier sistema desarrollado con arquitectura N-Capas.

Capa de presentación

Esta capa es responsable de mostrar información al usuario e interpretar sus acciones. Los componentes de las capas de presentación implementan la funcionalidad requerida para que los usuarios interactúen con la aplicación. Normalmente es recomendable subdividir dichos componentes en varias subcapas aplicando patrones de tipo *Model-View-Controller* (MVC). Las subcapas más comúnmente utilizadas son:

- Subcapa de Componentes Visuales (Vistas): contiene componentes que proporcionan el mecanismo base para que el usuario utilice la aplicación, por lo general controles visuales que muestran y reciben datos proporcionados por el usuario.
- Subcapa de Controladores: Para ayudar a sincronizar y orquestar las interacciones del usuario puede ser útil conducir el proceso utilizando componentes separados de los componentes propiamente gráficos. Esto impide que el flujo de proceso y lógica de gestión de estados esté programada dentro de los propios controles y formularios visuales y permite reutilizar dicha lógica y patrones desde otros interfaces o "vistas". También es muy útil para poder realizar pruebas unitarias de la lógica de presentación. Estos "*Controllers*" son típicos de los patrones MVC y derivados.

Capa de lógica de negocio

Esta capa es responsable de representar los conceptos del negocio, información sobre la situación de los procesos de negocio e implementación de las reglas del dominio, así como los estados que reflejan la situación de los procesos de negocio.

Los componentes de esta capa implementan la funcionalidad principal del sistema y encapsulan toda la lógica de negocio relevante. Básicamente suelen ser clases que implementan la lógica del dominio dentro de sus métodos. Siguiendo los patrones de arquitecturas N-Capas, esta capa tiene que ignorar completamente los detalles de persistencia de datos. Estas tareas de persistencia deben ser realizadas por la capa de integración y ser solo coordinadas por la capa de lógica de negocio.

Esta capa define los trabajos que la aplicación como tal debe realizar y dirige a los objetos del dominio y de infraestructura que son los que internamente deben resolver los problemas. También se implementa en esta capa la coordinación de la aplicación, como coordinación de transacciones, ejecución de unidades de trabajo, y en definitiva llamadas a tareas necesarias para la aplicación.

Capa de recursos

Esta capa proporciona la capacidad de persistir datos así como lógicamente acceder a ellos. Pueden ser datos propios del sistema o incluso acceder a datos expuestos por sistemas externos (Servicios Web externos, etc.). Así pues, esta capa de persistencia de datos expone el

acceso a datos a las capas superiores, normalmente las capas de lógica de negocio, esta exposición debe realizarse de una forma desacoplada.

Esta capa también proporciona capacidades técnicas genéricas que dan soporte a capas superiores. En definitiva, son “bloques de construcción” ligados a una tecnología concreta para desempeñar sus funciones. Existen muchas tareas implementadas en el código de una aplicación que se deben aplicar en diferentes capas. Estas tareas o aspectos transversales implementan tipos específicos de funcionalidad que pueden ser utilizados desde componentes de cualquier capa. Las funcionalidades transversales más comunes, son: Seguridad (Autenticación, Autorización y Validación) y tareas de gestión de operaciones (políticas, *logging*, trazas, monitorización, configuración, etc.).

3. Perfiles y Metamodelos

En esta sección se introducen y comparan dos de los mecanismos más utilizados en MDD: perfiles UML y metamodelos. Su análisis es fundamental, ya que a la hora definir la notación NMMp se dudó por representarla a través de un metamodelo MOF o mediante un perfil UML. Finalmente, se optó por esta solución. Para una descripción en profundidad de las tecnologías MDA puede consultarse (Fernandez and Navarro, 2009)

3.1 Perfiles frente a Metamodelos

Model-Driven Architecture (MDA) (OMG-MDA, 2003) es una iniciativa definida por *Object Management Group* (OMG) (OMG, 2010) que promueve el desarrollo de sistemas software basado en modelos. En MDA, los diferentes aspectos de un sistema software son representados a través de modelos que, en algún sentido, pueden ser vistos como evoluciones de un sistema desde la "realidad" hasta el código. Concibiendo "realidad" como abstracción y código como realización, los modelos parten desde representaciones abstractas a representaciones concretas de la aplicación.

MDA define tres modelos que representan diferentes vistas de un sistema (desde lo abstracto a lo concreto):

- *Modelo Independiente del Cómputo* (CIM), es un modelo en términos del dominio de la aplicación.
- *Modelo Independiente de la Plataforma* (PIM), es un modelo de la aplicación que omite cualquier detalle que lo relacione con plataformas específicas de implementación.
- *Modelo Específico de la Plataforma* (PSM), es un modelo PIM que incluye detalles específicos de una plataforma de implementación.

Dado que cada uno de estos modelos puede ser visto como una evolución del anterior, es deseable establecer un mecanismo que permita transformar automáticamente un modelo en el siguiente. OMG aporta la especificación del lenguaje *Query/View/Transformation* (QVT) (OMG-QVT, 2009) para hacer dichas transformaciones.

En (OMG-MDA, 2003) se presentan algunas opciones para definir las transformaciones entre modelos. Una de las más populares es definiendo reglas de transformación entre los

metamodelos que especifican los modelos que serán relacionados. Un metamodelo no es más que un modelo que puede ser usado para describir otros modelos. Por ejemplo, se pude usar el *Modelo Entidad-Relación* (E-R) (Pin-Shan Chen, 1976b) para describir diagramas de clases UML (OMG-UML, 20110), definiendo en el metamodelo E-R las entidades "Clase" y "Atributo", y la relación "ClaseTieneAtributo" entre ellas.

De igual forma, se puede utilizar los diagramas de clases UML como lenguaje de metamodelado (OMG-UML, 2010b). Sin embargo, los diagramas de clases UML tienen en sí mismos un metamodelo muy complejo y con semántica de clases orientadas a objetos. Por lo tanto, si se proporciona un metamodelo UML para describir modelos E-R y se define las clases "Entidad" y "Atributo" y la relación de agregación entre ellas, se podría pensar directamente en dos clases en Java (ó C++) llamadas "Entidad" y "Atributo" donde la primera tiene a la segunda como atributo.

Para resolver estas dificultades, OMG ha definido *UML Infrastructure* (OMG-UML, 2010a), que define un metamodelo núcleo que puede ser reutilizado por UML y MOF (*Meta-Object Facility*) (OMG-MOF, 2009). MOF es un lenguaje de metamodelado que básicamente proporciona una versión simplificada de diagramas de clase UML como lenguaje de metamodelado. *UML Infrastructure* define también cuatro niveles de metamodelado:

- M3: meta-metamodelos (por ejemplo MOF).
- M2: metamodelos (por ejemplo, un metamodelo UML desarrollado como una instancia de MOF).
- M1: Modelo (por ejemplo, un modelo UML de una aplicación particular desarrollada como una instancia de UML).
- M0: Instancias en tiempo de ejecución (por ejemplo, una instancia de una clase Java descrita en el modelo UML de una aplicación particular).

Además, *UML Infrastructure* incluye el concepto de *Perfil*, definido en el nivel M3. Los perfiles permiten enriquecer los metamodelos de nivel M2 definidos con MOF con *estereotipos*. Un estereotipo es una metaclase en el nivel M2 que puede ser relacionada con cualquier otra metaclase de su nivel. De esta forma, se puede incluir en M1 estereotipos como etiquetas dentro de las instancias de las metaclases de M2. Por ejemplo, en M2, el estereotipo "`input`" se puede relacionar con la metaclase "`Action`", y de esta forma en M1 se puede incluir la etiqueta

«input» en las acciones de los diagramas de actividad UML, como se muestra en (Navarro et al., 2008c).

MDA es el marco de trabajo proporcionado por OMG. Un término alternativo que sigue los principios MDA, pero independientemente de estándar MDA es *Model-Driven Development* (MDD), frecuentemente los términos MDA y MDD son utilizados indistintamente. MDD se refiere a la actividad que es llevada a cabo por los desarrolladores de software, mientras que MDA se utiliza para referirse a la definición formal de OMG, la cual está orientada en la especificación formal de un marco de trabajo en el cual MDD pueda operar (Gardner and Yusuf, 2006).

Es evidente que la aplicación de MDD requiere la utilización de metamodelos, en algunos casos los metamodelos pueden ser predefinidos por una organización, tal como el metamodelo de UML. En otros casos los metamodelos deben ser definidos para especificar un lenguaje particular, tal como *Web Modeling Language* (WebML) (Moreno et al., 2007). Finalmente, también existe la posibilidad de que metamodelos existentes puedan ser adaptados a dominios específicos a través de perfiles, tal como *UML Web Application Extension* (UML WAE) (Conallen, 1999).

Existe una regla de oro en la aplicación de MDD con respecto a los metamodelos:

1. Si existe un metamodelo válido entonces utilice este en el proceso MDD.
2. Si no existe un metamodelo válido entonces defínalo.
3. Si existe un metamodelo que podría ser válido si tuviera algunas etiquetas específicas entonces personalícelo a través de un perfil.

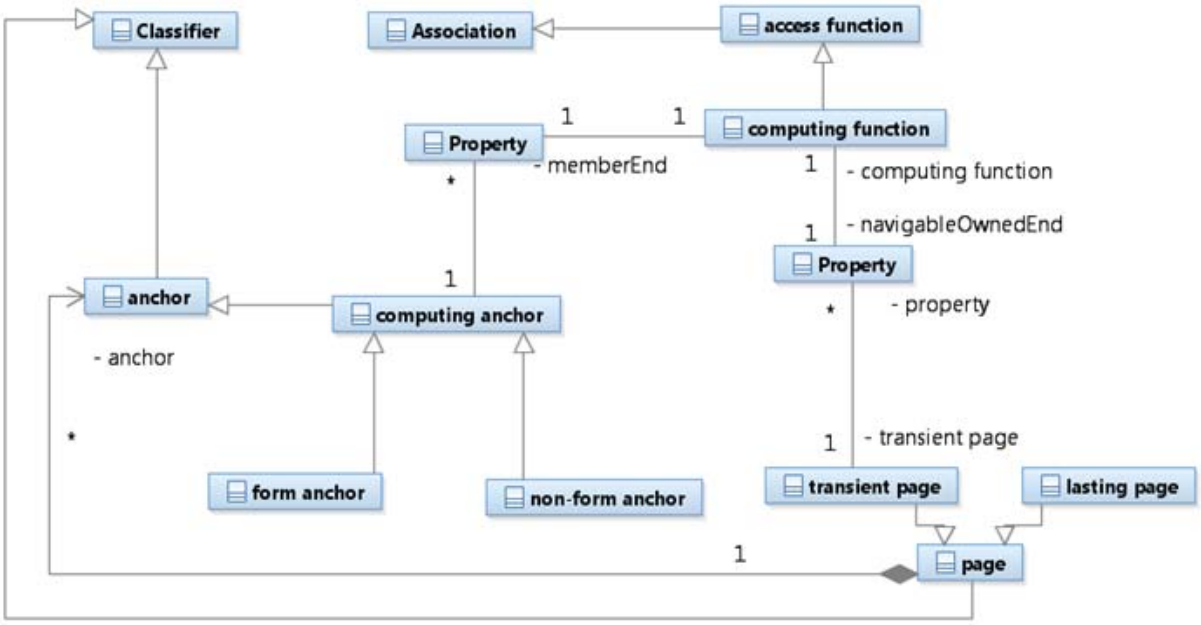
En muchos casos existen las opciones de, definir un nuevo metamodelo ó personalizar uno existente a través de perfiles. En (Brucker and Doser, 2007) se definen algunas reglas para seleccionar una u otra alternativa, sin embargo, estas reglas están centralizadas en las propiedades sintácticas de los lenguajes y no el los detalles prácticos de los dos enfoques.

A continuación se comentará la experiencia práctica en la especificación formal de los modelos NMM tanto a través de la definición de un metamodelo específico para la notación NMM como a través de la definición de un perfil para la personalización de un metamodelo existente, con el objetivo de aplicar el enfoque MDD en la transformación de los abstractos modelos NMM en detallados modelos UML-WAE.

3.1.1 Metamodelo MOF para NMM

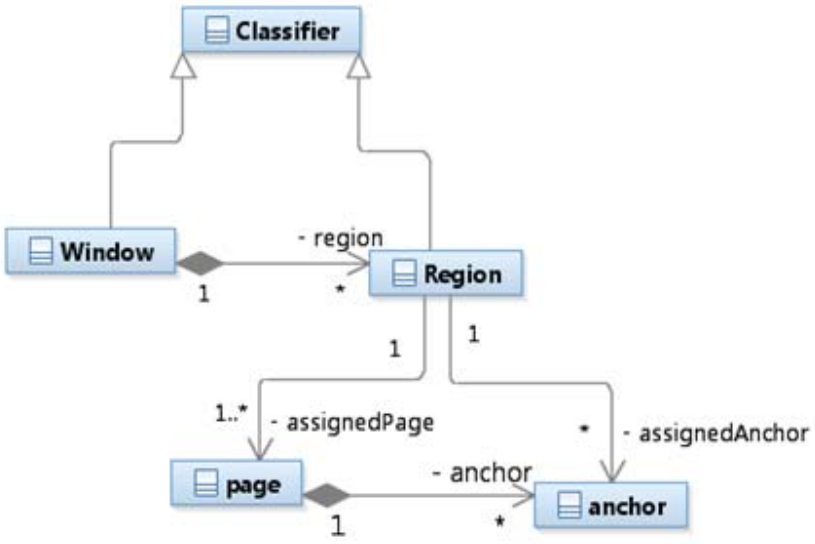
La primera opción a considerar es desarrollar un metamodelo MOF para la notación NMM. Por simplicidad, este metamodelo hace uso del paquete de UML `Core::Constructs`. La figura 3-1 muestra el metamodelo MOF para los diagrama de páginas NMM.

Figura 3-1 Metamodelo MOF para diagramas de página NMM



La figura 3-2 muestra el metamodelo MOF para los diagramas de región y mezcla NMM. Es necesario comentar que los metamodelos de la figura 3-1 y figura 3-2 pueden ser considerados como una extensión del metamodelo MOF de UML.

Figura 3-2 Metamodelo MOF para diagramas de región y diagramas de mezcla NMM



Estos metamodelos proporcionan una descripción compacta de la sintaxis abstracta de la notación NMM. Para ser usados en el contexto de una herramienta CASE, la alternativa más razonable parece ser *Eclipse Graphical Modeling Framework* (GMF) (Eclipse, 2011a). Para definir una herramienta CASE para la notación NMM basada en estas metamodelos hay que tener en cuenta los siguientes tres pasos:

- Proporcionar una definición gráfica, lo cual significa definir el aspecto visual del editor a generar.
- Proporcionar una definición de herramientas, lo cual comprende aspectos relacionados con el editor visual como definición de barra de herramientas, menús, etc.
- Definir la relación entre la lógica de negocio (el modelo de dominio de Ecore) y los modelos visuales (definición gráfica y definición de herramientas).

Los metamodelos NMM definidos en la figura 3-1 y figura 3-2 fueron generados usando la herramienta CASE *IBM Rational Software Architect* (IBM, 2010). Entre las ventajas de usar la opción de definir un nuevo metamodelo para NMM podemos mencionar:

- El poder descriptivo del metamodelo MOF de NMM que permite la definición de una nueva notación personalizada.

Entre los inconvenientes podemos comentar:

- La necesidad de conocer un lenguaje de metamodelado (por ejemplo MOF o Ecore). Está claro que estos lenguajes se asemejan mucho a los diagramas de clases UML, pero el uso de UML para modelar algo distinto de clases orientadas a objetos no es trivial.
- La necesidad de utilizar dos herramientas:
    - Eclipse Ecore u otra herramienta CASE.
    - Framework de modelado gráfico de Eclipse (*Eclipse Graphical Modeling Framewok (GMF)*).
- Dependencia en dos framework diferentes que no tienen soporte ni están integrados por ninguna herramienta CASE comercial:
    - Framework de modelado Eclipse (*Eclipse Modeling Framework (EMF)*).
    - Framework de modelado gráfico de Eclipse (*Eclipse Graphical Modeling Framewok (GMF)*).
- La falta de soporte para la administración de modelos NMM por una herramienta UML CASE de propósito general.

3.1.2 Perfil UML para NMM - NMMp

La segunda opción a considerar es definir un perfil UML para especificar la notación NMM. Existen varias herramientas CASE que soportan la definición de perfiles UML, para este trabajo se ha seleccionado la herramienta CASE *Borland Together 2008* (Borland, 2011) por incluir no solo un entorno para la definición de perfiles UML sino todo un conjunto de capacidades MDA basadas en los estándares de modelado OCL, QVT y MOF.

La figura 3-3 muestra el perfil UML para los diagramas de página y diagramas de regiones NMM. No es necesario definir un perfil UML para los diagramas de mezcla porque, como lo comentamos a continuación, este tipo de diagrama está definido implícitamente en el perfil UML de los diagramas de página. En la figura 3-3 el perfil NMM, llamado de aquí en adelante NMMp, se muestra con la notación de perfiles usada por Borland Together en lugar de la notación gráfica propuesta en UML Infrastructure. La idea es mostrar el poder expresivo de la notación usada por la herramienta. Otras notaciones usadas por otras herramientas como *IBM Rational Software Architect* son similares.

No es necesario definir un perfil UML para los diagramas de mezcla porque la función de asignación de páginas por defecto a las regiones se hace a través del valor etiquetado

"DefaultPage" definido en el estereotipo «Region», que en la notación de Borland Together mostrada en la figura 3-3 se representa como un atributo de dicho estereotipo. De igual forma la función de asignación de la región destino a las anclas NMM se ha implementado asociando un valor etiquetado llamado "Target" a los elementos NMM «Retrieval Anchor», «Form Computing Anchor» y «Non Form Computing Anchor»; el valor etiquetado no se ha relacionado directamente en el estereotipo «Anchor» por detalles técnicos relacionados con la definición de las transformaciones QVT *Operational-mappings* ejecutadas por la herramienta CASE.

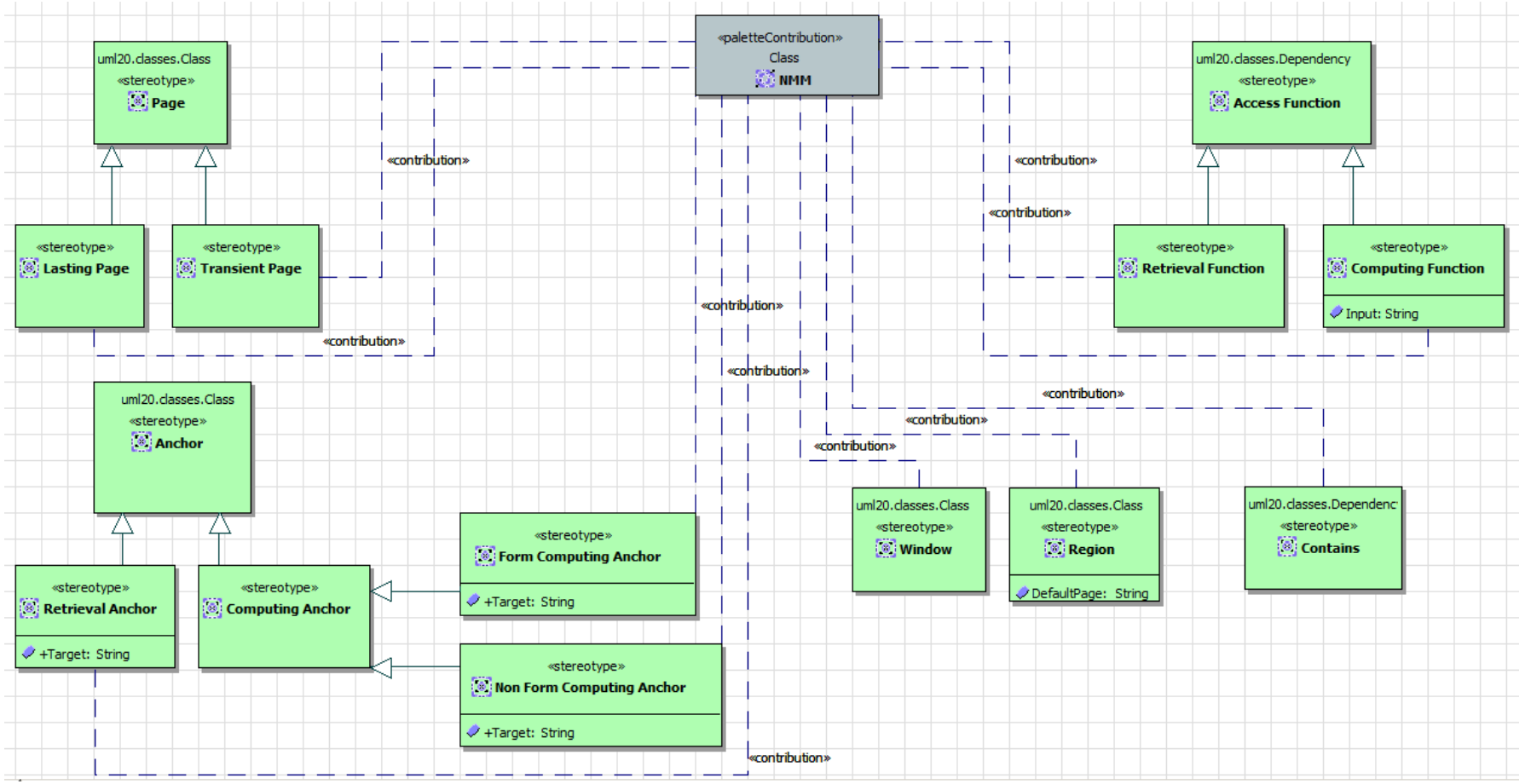
Entre las ventajas de usar la opción de definir un Perfil UML para NMM podemos mencionar:

- No es necesario un conocimiento profundo sobre lenguajes de metamodelado para definir un perfil UML.
- La disponibilidad de una herramienta CASE (Borland Together) que permite la definición del perfil, su uso y la ejecución de transformaciones QVT en el mismo entorno.
- Integración de modelos UML personalizados a través de un perfil con otros modelos UML para describir diseños más detallados, por ejemplo, utilizar modelos con perfil UML-WAE para describir la capa de presentación de una aplicación Web y modelos UML estándar para describir el resto de las capas de la aplicación.

Entre los inconvenientes podemos comentar:

- Falta de poder descriptivo de la notación de los perfiles usada por la herramienta CASE.
- El coste de la licencia de la herramienta CASE Borland Together.

Figura 3-3 Perfil NMMp : perfil UML para diagramas de página y diagramas de región NMM definido en Borland Together



Como hemos visto, en general, resulta difícil decidir cuándo debemos crear un nuevo metamodelo y cuándo en cambio es mejor definir una extensión de UML usando los mecanismos de extensión estándares definidos con ese propósito. Cada una de estas dos alternativas presenta ventajas e inconvenientes. Así, definir un nuevo lenguaje ad-hoc permite mayor grado de expresividad y correspondencia con los conceptos del dominio de aplicación particular. Sin embargo, y aún cuando esos nuevos lenguajes ad-hoc se describan con MOF, el hecho de no respetar el metamodelo estándar de UML va a impedir que las herramientas UML existentes en el mercado puedan manejar sus conceptos de una forma natural.

Una de las características más importantes de la notación NMM es que permite modelar aplicaciones Web sin tener en cuenta en principio sus detalles arquitectónicos y sobre todo independientemente de cualquier herramienta. Las aplicaciones modeladas con NMM no deberían estar ligadas a ninguna herramienta en particular, por lo tanto lo más razonable parece ser optar por definir un perfil UML para que los modelos NMM puedan ser gestionados por cualquier herramienta que soporte el estándar UML.

3.2 Perfil UML-WAE

UML Web Application Extension (Conallen, 1999) es un perfil UML que permite modelar los elementos particulares de las arquitecturas Web integrados con otros elementos modelados con UML estándar logrando obtener una visión más completa de una aplicación Web. El paquete Profiles de UML 2.0 define una serie de mecanismos para extender y adaptar las metaclases de un metamodelo cualquiera a las necesidades concretas del dominio de una aplicación. Un perfil se define en un paquete UML estereotipado con `«profile»`, que extiende a un metamodelo o a otro perfil. Tres son los mecanismos que se utilizan para definir perfiles: estereotipos (*stereotypes*), restricciones (*constraints*) y valores etiquetados (*tagged values*).

Los *estereotipos* permiten extender el vocabulario del lenguaje añadiendo un nuevo significado semántico a los elementos de un modelo. Los *valores etiquetados* permiten extender las propiedades de los elementos del modelo. Las *restricciones* son extensiones a la semántica del lenguaje que especifican las condiciones bajo las cuales un modelo puede ser considerado "*bien formado*".

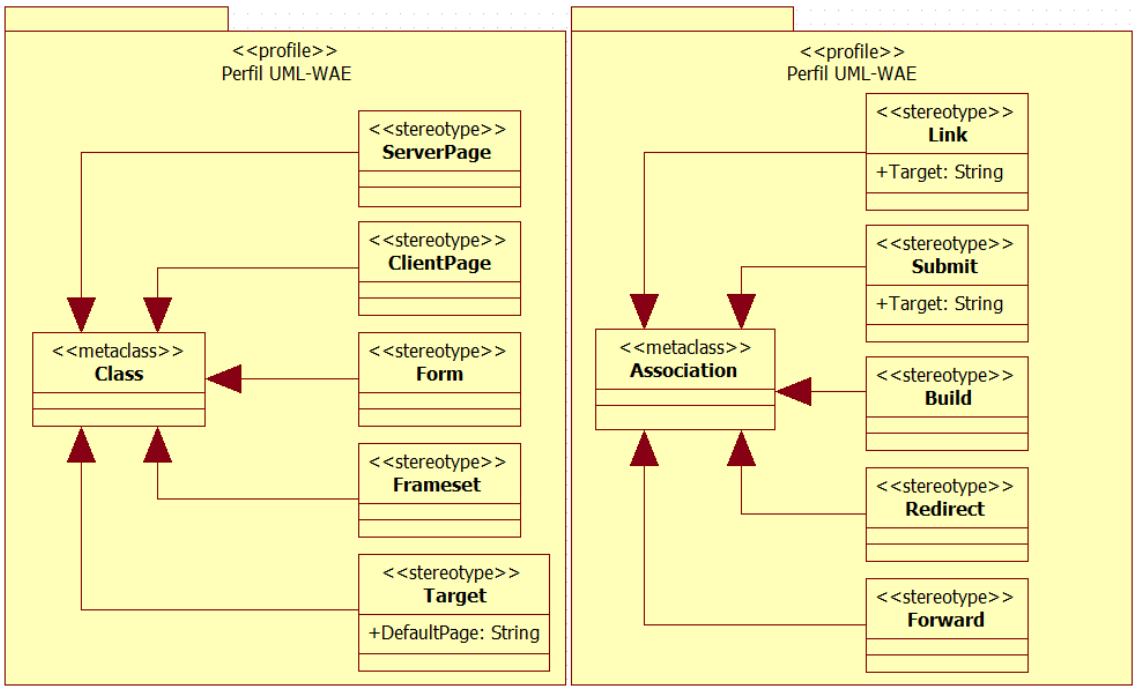
El perfil UML-WAE se expresa en términos de estereotipos, valores etiquetados y restricciones que extienden la notación de UML. Estos elementos permiten definir nuevos tipos de constructores que pueden ser usados para modelar aplicaciones Web.

En la figura 3-4 se muestra los estereotipos del perfil UML-WAE que son relevantes en este trabajo utilizando la notación gráfica propuesta en *UML Infraestructure*.

Como puede observarse en la figura 3-4, cada estereotipo se define gráficamente dentro de una caja estereotipada con `«stereotype»`, tienen un nombre y se asocian a un conjunto de elementos del metamodelo sobre los que puede aplicarse. Tal y como se indica, en el perfil UML-WAE las clases y las asociaciones de UML son adaptadas al dominio Web.

Un valor etiquetado es un metaatributo que se asocia a una metaclase del metamodelo extendido por un perfil a través de un estereotipo. Todo valor etiquetado debe tener un nombre y un tipo, se representan de forma gráfica como atributos de la clase que define el estereotipo. Como se muestra en la figura 3-4, para los fines de este trabajo se ha agregado a los estereotipos `«Link»` y `«Submit»` un valor etiquetado denominado "`Target`" de tipo `String` que indica la región destino de las páginas que reciben esta asociación. De igual forma al estereotipo `«Target»` se ha agregado el valor etiquetado "`DefaultPage`" también de tipo `String` que indica la página por defecto asignada a cada clase estereotipada con `«Target»`.

Figura 3-4 Perfil UML-WAE en notación gráfica de UML Infraestructure



A continuación se describe los estereotipos, valores etiquetados y restricciones definidos en el perfil UML-WAE y que se muestran el la figura 3-4. En las tablas 3-1, 3-2, 3-3, 3-4 y 3-5 se describen los estereotipos que se aplican a clases y en la tablas 3-6, 3-7, 3-8, 3-9 y 3-10 los estereotipos que se aplican a asociaciones.

Tabla 3-1 Estereotipo «ServerPage» para página de servidor

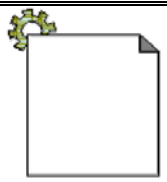
Nombre	Página de servidor «ServerPage»
Clase metamodelo	Clase
Descripción	Una página de servidor representa una página Web que tiene scripts que son ejecutadas por el servidor. Estos scripts operan con recursos en el servidor (bases de datos, lógica de negocio, sistemas externos, etc). Los métodos de la clase representan las funciones en el script, y sus atributos representan las variables que son visibles en el alcance de la página (accesible por todas las funciones en la página).
Icono	
Restricciones	Las páginas del servidor pueden tener sólo relaciones con objetos en el servidor.
Valores etiquetados	Artefacto de scripting ó lenguaje ó artefacto que deben ser usado para ejecutar ó interpretar esta página (ASP, JSP, servlets, Perl, etc.)

Tabla 3-2 Estereotipo «ClientPage» para página de cliente

Nombre	Página de cliente «ClientPage»
Clase metamodelo	Clase
Descripción	Una página de cliente se corresponde con una página Web estructurada con un lenguaje de marcado como HTML ó XML que contiene una mezcla de datos y presentación. Las páginas del cliente son presentadas por navegadores Web y pueden contener scripts que son interpretados y ejecutados por el

	navegador. Las páginas del cliente pueden asociarse con otras páginas del cliente o del servidor.
Icono	
Restricciones	Ninguna
Valores etiquetados	TitleTag – El título de la página desplegado por el navegador. BaseTag – La URL base para obtener la página. BodyTag – El conjunto de atributos para la etiqueta <body> que establece sus valores por defecto.

Tabla 3-3 Estereotipo «Form» para formulario

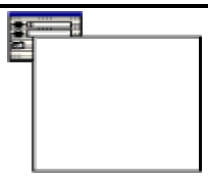
Nombre	Formulario «Form»
Clase metamodelo	Clase
Descripción	Una clase estereotipada con «Form» es una colección de campos de entrada que son parte de una página del cliente. Esta clase mapea directamente a la etiqueta <form> de HTML. Sus atributos representan los campos de la entrada del formulario (cajas de texto, áreas de texto, botones de selección, campos ocultos, etc.). Una clase estereotipada con «Form» no puede tener operaciones, cualquier operación que interactúe con el formulario debe estar definida en la página cliente que contenga al formulario.
Icono	
Restricciones	Ninguna
Valores etiquetados	Método usado para enviar los datos a la URL, puede ser GET ó POST

Tabla 3-4 Estereotipo «Frameset»

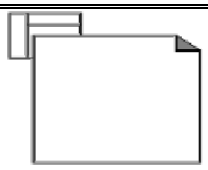
Nombre	Frameset «Frameset»
Clase metamodelo	Clase
Descripción	Esta clase se corresponde directamente a la etiqueta <frameset> de HTML. Esta página divide la interfaz de usuario en regiones rectangulares, en cada una de las cuales se puede presentar una página de cliente diferente.
Icono	
Restricciones	Debe contener al menos una clase estereotipada «ClienPage» ó «Target»
Valores etiquetados	Filas (Rows): número de filas en las cuales la interfaz es dividida. Columnas (Cols): número de columnas en las cuales la interfaz es dividida.

Tabla 3-5 Estereotipo «Target»

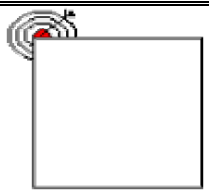
Nombre	Target «Target»
Clase metamodelo	Clase
Descripción	Una clase que representa una región con nombre dentro de un frameset
Icono	
Restricciones	El nombre de una clase estereotipada «Target» debe ser único para cada cliente del sistema.
Valores etiquetados	Fila (Row): el número de fila en el cual la región debe ser dibujada. Columna (Col): el número de columna en el cual la región debe ser dibujada.

Tabla 3-6 Estereotipo «Link»

Nombre	Link «Link»
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde una página cliente a otra página cliente o pagina de servidor, se corresponde directamente con la etiqueta <a> de HTML.
Restricciones	Ninguna
Valores etiquetados	Parámetros: contiene los parámetros pasados con la petición http. Este valor etiquetado puede tener formato de cadena y estar codificado.

Tabla 3-7 Estereotipo «Build»

Nombre	Build «Build»
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde una página de servidor hasta una página cliente. Esta relación identifica el resultado HTML o XML de la ejecución de una página de servidor. Una página de servidor puede construir muchas páginas cliente, pero una página cliente puede ser construida solo por una página de servidor.
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Tabla 3-8 Estereotipo «Submit»

Nombre	Submit «Submit»
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida entre una clase estereotipada con «Form» y una página de servidor. La página de servidor procesa los datos que la página estereotipada con «Form» le envía, que por lo general son los campos de la entrada del formulario.
Restricciones	Ninguno.

Valores etiquetados	Ninguno.

Tabla 3-9 Estereotipo «Forward»

Nombre	Forward «Forward»
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde una página de servidor a otra página de servidor ó página cliente simbolizando la delegación del procesamiento de una petición de un cliente.
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Tabla 3-10 Estereotipo «Redirect»

Nombre	Redirect «Redirect»
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde una página de cliente ó una página de servidor a otra página cualquiera. Esta asociación indica un comando para que el cliente solicite otro recurso.
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Como se comentó antes, para este trabajo se ha seleccionado la herramienta CASE *Borland Together 2008* por incluir no solo un entorno para la definición de perfiles UML sino todo un conjunto de capacidades MDA. Borland Together 2008 proporciona un tipo especial de proyecto llamado "*Profile Definition Project*" diseñado para crear nuevas definiciones de perfiles UML. Cuando se crea un nuevo proyecto para la definición de perfiles es necesario especificar el metamodelo que se desea extender, Borland Together 2008 permite extender los siguientes metamodelos:

- BPMN.
- ER Physical.
- UML 1.4.
- UML 2.0.

En un proyecto para la definición de perfiles los siguientes elementos están disponibles en la barra de herramientas del diagrama:

- Estereotipo ("*Stereotype*"): permite agregar nuevos estereotipos en el proyecto como clases con la etiqueta «stereotype». Los estereotipos entre muchas otras contienen las propiedades "*Extended Metaclass*" e "*Icon*". En la propiedad "*Extended Metaclass*" se establece la metaclase que será extendida por el estereotipo. La propiedad "*Icon*" permite asociarle un icono gráfico al estereotipo.
- Paleta de contribución ("*Palette Contribution*"): permite crear una paleta de herramientas con los estereotipos definidos en el proyecto.
- Enlace de contribución ("*Contribution Link*"): Conecta los estereotipos definidos en el proyecto con la paleta de contribución.

En la figura 3-5 se muestra la implementación del perfil UML-WAE que extiende el metamodelo de UML 2.0 en Borland Together 2008 para este trabajo. Notar que en la representación gráfica de cada estereotipo se muestra la metaclase que esta siendo extendida y que los valores etiquetados se representan como atributos de los estereotipos.

En la figura 3-6 se muestra un proyecto al cual se le ha *aplicado* el perfil UML-WAE de la figura 3-5. Notar la barra de herramientas del lado izquierdo que contiene acceso directo a los estereotipos definidos y asociados a la paleta de contribución. Todas las clases estereotipadas tanto en el diagrama como en la barra de herramientas presentan el icono que se les ha asociado en la definición del perfil.

Figura 3-5 Definición del perfil UML-WAE en *Borland Together 2008*

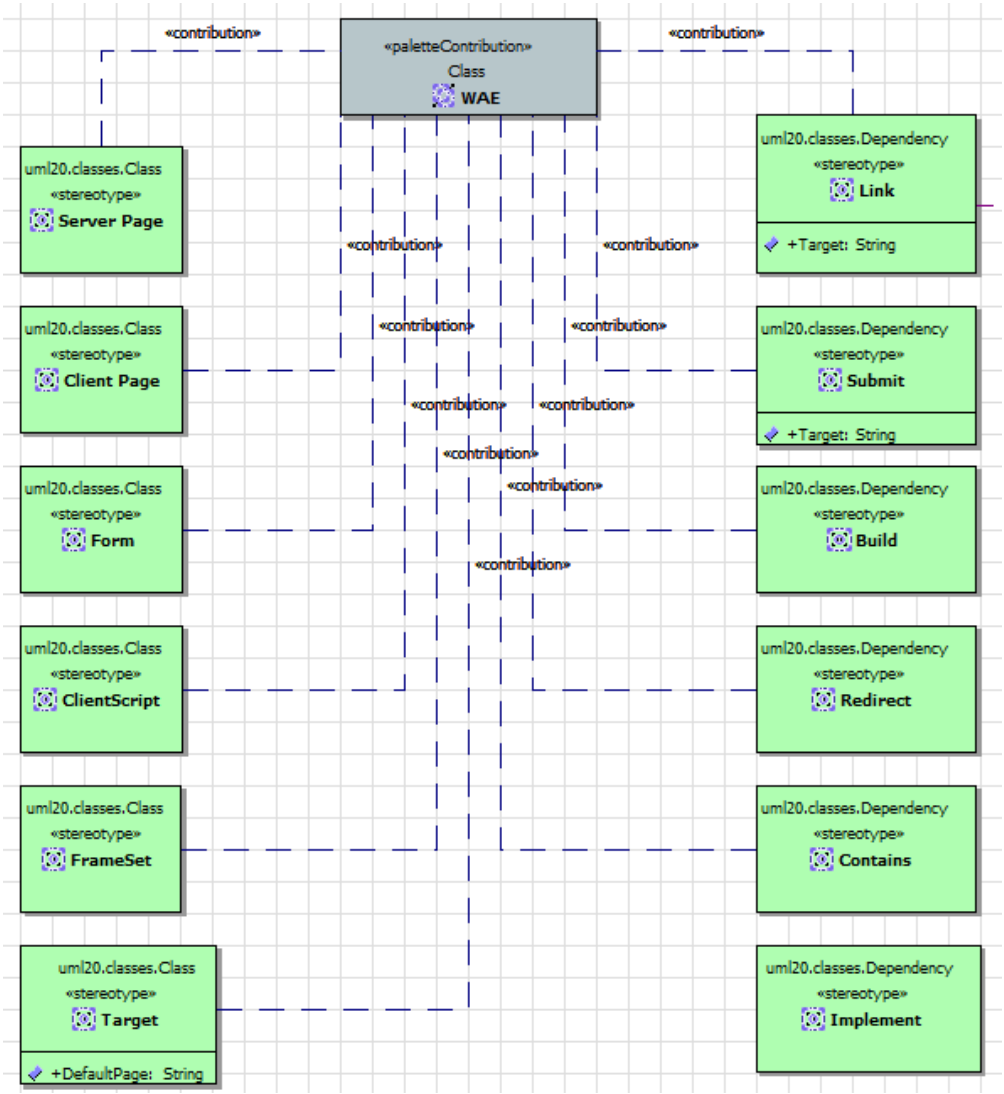
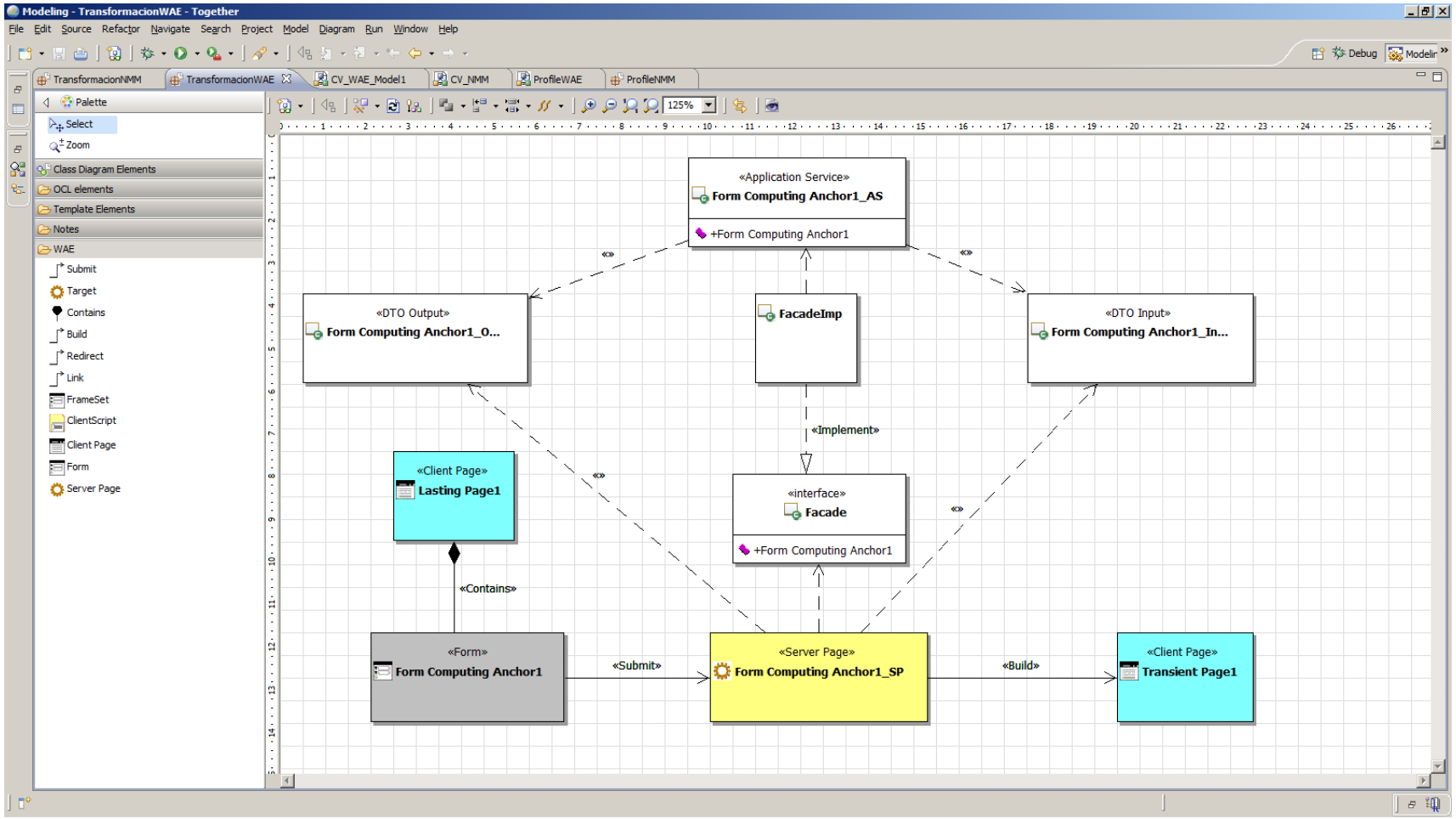


Figura 3-6 Modelo personalizado con el perfil UML-WAE en *Borland Together 2008*



3.3 Perfil NMMp

La notación NMM (Navarro et al., 2008b) permite modelar la capa de presentación de las aplicaciones Web teniendo en cuenta las arquitecturas multicapas pero omitiendo en principio cualquier detalle arquitectónico ó especifico de alguna plataforma de implementación. Los elementos que hacen parte de la notación NMM además de su caracterización formal, poseen una representación visual que simplifica su uso y que ha sido utilizada en el desarrollo del perfil NMMp.

Al igual que el perfil UML-WAE comentado anteriormente, el perfil NMMp se ha desarrollado en términos de estereotipos, valores etiquetados y restricciones que extienden el metamodelo de UML 2.0 permitiendo definir nuevos tipos de constructores para modelar la capa de presentación de aplicaciones Web. Entre los objetivos que se persiguen con el desarrollo del perfil NMMp podemos comentar:

- Disponer de una terminología y vocabulario propio del dominio Web (utilizar términos propios dentro del proceso de modelado de una aplicación Web como “página estática”, “página dinámica”, “anclas de recuperación”, “anclas computacionales”, etc.).
- Definir una nueva notación para símbolos ya existentes más acorde con el dominio Web (utilizar representación gráfica más adecuada para los componentes Web).
- Añadir cierta semántica que no aparece determinada de forma precisa en el metamodelo de UML 2.0 (por ejemplo, que las páginas dinámicas son construidas a través de procesos computacionales).
- Añadir cierta semántica que no existe en el metamodelo de UML 2.0 (por ejemplo que las anclas de recuperación solo dan acceso a las páginas estáticas).
- Añadir información que puede ser útil en el proceso de transformación de un modelo con el perfil NMMp a un modelo con perfil UML-WAE.

UML Infraestructure se limita a definir el concepto de perfil y sus contenidos; sin embargo en (Fuentes and Vallecillo, 2004) se exponen algunas guías y recomendaciones prácticas que se han intentado seguir para desarrollar el perfil NMMp:

1. Definir el metamodelo del dominio Web en términos de la notación NMM utilizando los mecanismos del propio UML ó MOF (clases, relaciones de herencia, asociaciones, etc.) de la forma usual como se haría si el objetivo no fuese definir un perfil UML. Se debe incluir la definición de los elementos propios de la notación NMM, las relaciones entre ellos, así como las restricciones que limitan el uso de estas entidades y de sus relaciones, tal como se muestra en la sección *Metamodelo MOF para NMM* de este mismo Capitulo.
2. Basados en el metamodelo del dominio Web en términos de la notación NMM, se define el perfil NMMp. Dentro del paquete `«profile»` se incluye un estereotipo por cada uno de los elementos del metamodelo que deseamos incluir en el perfil. Estos estereotipos tendrán el mismo nombre que los elementos del metamodelo, estableciéndose de esta forma una relación entre el metamodelo y el perfil como se muestra en la figura 3-7 y figura 3-8.
3. Es importante tener claro cuáles son los elementos del metamodelo de UML que estamos extendiendo sobre los que es posible aplicar un estereotipo. Ejemplo de tales elementos son las clases, asociaciones, atributos, operaciones, transiciones, paquetes, etc. El perfil NMMp extiende las asociaciones de UML como se muestra en la figura 3-7 y a las clases de UML como se muestra en la figura 3-8.
4. Definir como valores etiquetados de los elementos del perfil NMMp los atributos que aparezcan en el metamodelo. Incluir la definición de sus tipos, y sus posibles valores iniciales.
5. Definir las restricciones que forman parte del perfil NMMp a partir de las restricciones del dominio. Por ejemplo, las multiplicidades de las asociaciones que aparecen en el metamodelo del dominio, o las propias reglas de negocio de la aplicación deben traducirse en la definición de las correspondientes restricciones.

Figura 3-7 Perfil NMMp en notación gráfica de UML Infraestructure que extiende las asociaciones de UML

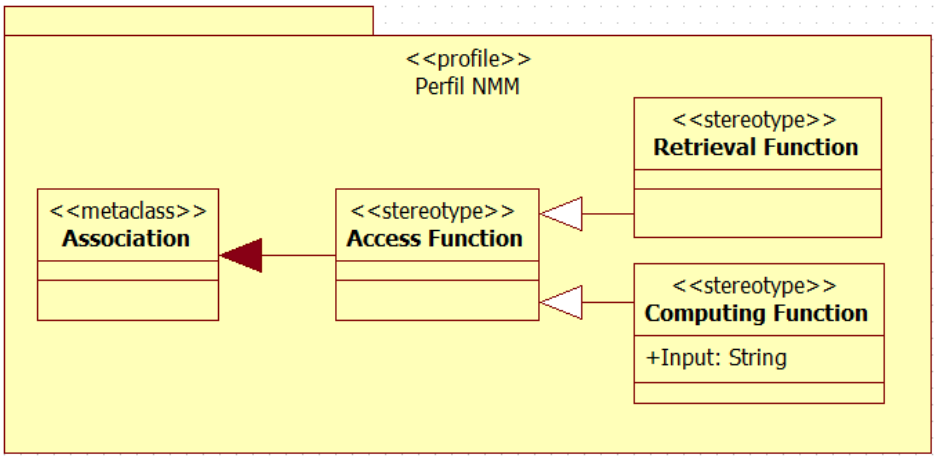
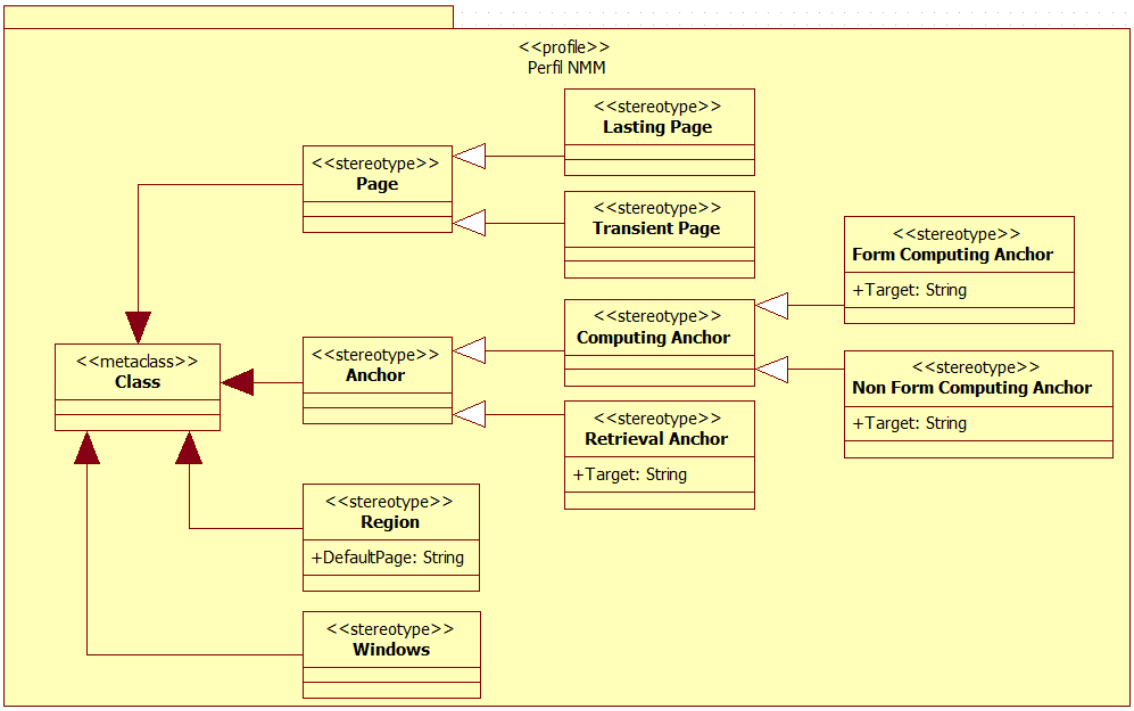


Figura 3-8 Perfil NMMp en notación gráfica de UML Infraestructure que extiende las clases de UML



No es necesario definir un perfil NMMp para los diagramas de mezcla NMM, porque como se muestra en la figura 3-8, se ha agregado a los estereotipos `«Retrieval Anchor»`, `«Form Computing Anchor»` y `«Non Form Computing Anchor»` un valor etiquetado denominado "`Target`" de tipo `String` que implementa la función de asignación de la región destino a las anclas NMM. De igual forma al estereotipo `«Region»` se ha agregado el valor etiquetado "`DefaultPage`" también de tipo `String`

que implementa la función de asignación de páginas por defecto a las regiones representadas con clases estereotipadas con «Region».

A continuación se describe los estereotipos, valores etiquetados y restricciones definidos en el perfil NMMp y que se muestran el la figura 3-1 y figura 3-2. En las tablas 3-11, 3-12, 3-13, 3-14, 3-15, 3-16 y 3-17 se describen los estereotipos que se aplican a clases y en la tablas 3-18 y 3-19 los estereotipos que se aplican a asociaciones.

Tabla 3-11 Estereotipo «Lasting Page» para página estática

Nombre	Página estática «Lasting Page»
Clase metamodelo	Clase
Descripción	Páginas estáticas son aquellas páginas que están completamente definidas antes de cualquier interacción con la aplicación, es decir no requieren de procesos computacionales para su elaboración.
Icono	
Restricciones	Las páginas estáticas solo pueden contener código en algún lenguaje de marcado como HTML ó XML y código Java Script.
Valores etiquetados	Ninguno.

Tabla 3-12 Estereotipo «Transient Page» para página dinámica

Nombre	Página dinámicamente creada «Transient Page»
Clase metamodelo	Clase
Descripción	Páginas dinámicas son aquellas que son construidas a través de componentes computacionales invocados por el servidor Web, es decir son generadas por medio de procesos computacionales lanzados como resultado de la interacción con la aplicación Web.
Icono	

Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Tabla 3-13 Estereotipo «`Retrieval Anchor`» para ancla de recuperación

Nombre	Ancla de recuperación «`Retrieval Anchor`»
Clase metamodelo	Clase
Descripción	Las anclas de recuperación son dispositivos dentro de una página Web que tiene la capacidad de iniciar una petición que retorna páginas estáticas.
Icono	
Restricciones	Ninguno.
Valores etiquetados	Target contiene la región destino por defecto asignada a cada ancla.

Tabla 3-14 Estereotipo «`Form Computing Anchor`» para ancla computacional en formulario

Nombre	Ancla computacional en formulario «`Form Computing Anchor`»
Clase metamodelo	Clase
Descripción	Las anclas computacionales en formulario representan botones de envío dentro de *Web forms* que tiene la capacidad de iniciar una petición que retorna páginas dinámicamente creada.
Icono	
Restricciones	Ninguno.
Valores etiquetados	Target contiene la región destino por defecto asignada a cada ancla.

Tabla 3-15 Estereotipo «Non Form Computing Anchor» para ancla computacional fuera de formulario

Nombre	Ancla computacional fuera de formulario «Non Form Computing Anchor»
Clase metamodelo	Clase
Descripción	Las anclas computacionales fuera de formulario representan dispositivos que generan peticiones desde cualquier proceso computacional diferente a los botones de envío
Icono	
Restricciones	Ninguno.
Valores etiquetados	Target contiene la región destino por defecto asignada a cada ancla.

Tabla 3-16 Estereotipo «Window» para ventana

Nombre	Ventana «Window»
Clase metamodelo	Clase
Descripción	Representa cada una de las ventanas en la interfaz gráfica de usuario de la aplicación Web.
Icono	
Restricciones	Ninguno.
Valores etiquetados	Ninguno.

Tabla 3-17 Estereotipo «Region» para región

Nombre	Region «Region»
Clase metamodelo	Clase
Descripción	Representa cada una de las regiones usadas por las ventanas de interfaz gráfica de usuario de la aplicación Web.

Icono	
Restricciones	Ninguno.
Valores etiquetados	DefaultPage representa la página por defecto asignada a la región.

Tabla 3-18 Estereotipo «`Retrieval Function`» para función de recuperación

Nombre	Función de recuperación «`Retrieval Function`»
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde un ancla de recuperación a una página estática.
Restricciones	Ninguna
Valores etiquetados	Ninguno.

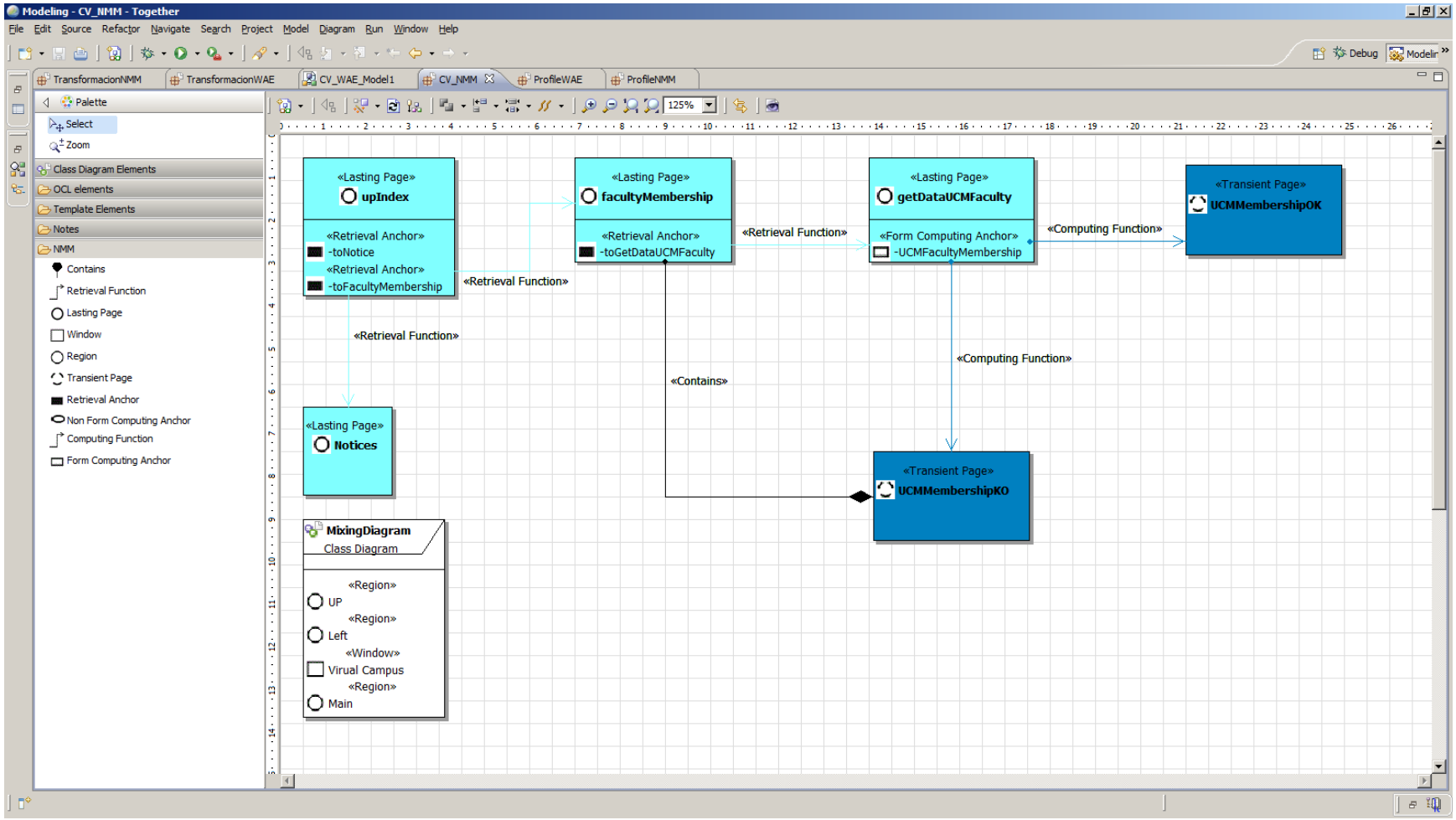
Tabla 3-19 Estereotipo «`Computing Function`» para función computacional

Nombre	Función Computacional «`Computing Function`»
Clase metamodelo	Asociación
Descripción	Representa una relación dirigida desde un ancla computacional a una página dinámica.
Restricciones	Ninguna
Valores etiquetados	Input contiene los parámetros necesarios par construir la página dinámica. Este valor etiquetado puede tener formato de cadena y estar codificado.

En la sección *Perfil UML para NMM* de este mismo Capitulo se ha comentado la implementación del perfil NMMp que extiende el metamodelo de UML 2.0 en Borland Together 2008. Cabe destacar que en la representación gráfica de cada estereotipo se muestra la metaclase que esta siendo extendida y que los valores etiquetados se representan como atributos de los estereotipos. En la figura 3-9 se muestra un proyecto al cual se le ha *aplicado* el perfil NMMp de la figura 3-3. Nótese la barra de herramientas del lado izquierdo que contiene acceso directo a los estereotipos

definidos y asociados a la paleta de contribución. Todas las clases estereotipadas tanto en el diagrama como en la barra de herramientas presentan el icono que se les ha asociado en la definición del perfil.

Figura 3-9 Modelo personalizado con el perfil NMMp en *Borland Together 2008*



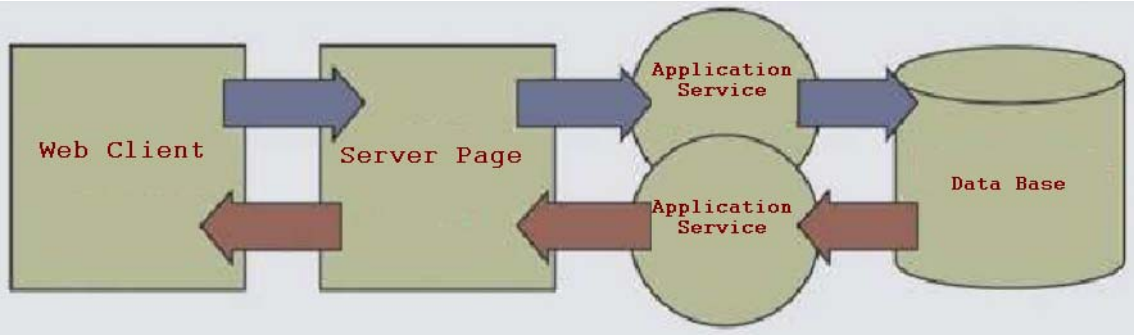
4. De NMM a UML-WAE Model 1

Esta sección define las transformaciones QVT Operational Mappings para traducir PIMs independientes de la arquitectura NMMp en PIMs UML WAE con una arquitectura de presentación Model 1.

4.1 Arquitectura Model 1

Model 1 (Brown et al., 2002; Brown et al., 2005) es un diseño arquitectónico para aplicaciones Web "centrado en las páginas" que utiliza el enfoque cliente/servidor. Este modelo básicamente elimina la presencia del controlador, permitiendo que las páginas Web del lado del servidor ("*Server Page*"), casi siempre encargadas de tareas de presentación, puedan acceder directamente a las capas intermedias de una aplicación Web para componer la respuesta a una petición de un cliente Web. La página Web del lado del servidor ("*Server Page*") es la encargada de recibir la petición, coordinar el procesamiento, componer y enviar la respuesta al cliente, como se muestra en la figura 4-1.

Figura 4-1 Arquitectura Model 1 ó arquitectura "*Page-Centric*"



La ventaja de este modelo es que es simple de implementar, permite al desarrollador de las páginas generar fácilmente el contenido dinámico basado en la petición del cliente y el estado de los recursos. Usualmente las "*Server Page*" usan alguna forma de *Modelo* para encapsular la lógica de negocio de la aplicación permitiendo su reutilización y al mismo tiempo minimizando al máximo la cantidad de código de procesamiento en las "*Server Page*". Sin embargo, la arquitectura Model 1 permite que grandes cantidades de código sean embebidas dentro de las "*Server Page*",

dificultando su mantenibilidad y mezclando conceptos de la lógica de negocio con presentación.

La arquitectura Model 1 es adecuada para aplicaciones pequeñas con pocas páginas y baja complejidad, sin embargo puede presentar problemas cuando es aplicada a grandes ó complejas aplicaciones ó para un número grande de clientes simultáneos dado que se debería procesar una cantidad significante de peticiones, y cada petición debería compartir la conexión a recursos potencialmente caros o escasos. Algunas de las debilidades de esta arquitectura son:

- Problemas de mantenibilidad: dado que las "Server Page" son responsables de administrar completamente las peticiones de los clientes, deben interactuar directamente con la capa de negocio. Esto puede ocasionar que la estructura de la aplicación termine siendo incorporada dentro de las mismas "Server Page". Esto obviamente puede hacer que las "Server Page" sean complicadas, con grandes cantidades de código y en últimas difíciles de mantener.
- Problemas de reusabilidad: cuando se incorpora procesamiento de lógica en las "*Server Page*", sin estar encapsulada en clases, esta lógica se vuelve difícil de reutilizar, obligando a los desarrolladores a utilizar la práctica del "*copy/paste*". Esta práctica no es solo mala desde el punto de vista de la reusabilidad sino que además introduce errores y disminuye la productividad. Aunque puede ser solucionada si el código se encapsula en clases.
- Problemas de seguridad: dado que cada "*Server Page*" es responsable de administrar las peticiones de los clientes y dado que algunas acciones de los clientes requieren que los usuarios estén autenticados/autorizados para acceder a recursos sensibles, es necesario que cada "*Server Page*" encapsule o coordine la lógica necesaria para permitir ó denegar el acceso a dichos recursos. Esto puede convertirse en un *agujero* de seguridad dado que las buenas prácticas recomiendan proporcionar los controles de seguridad por medio de un punto de acceso centralizado.

4.2 Transformación

Como se ha comentado anteriormente en este trabajo, los modelos NMM pueden ser vistos como versiones de alto nivel de los modelos UML-WAE donde los detalles arquitectónicos son omitidos, por lo cual, los modelos NMM pueden ser fácilmente

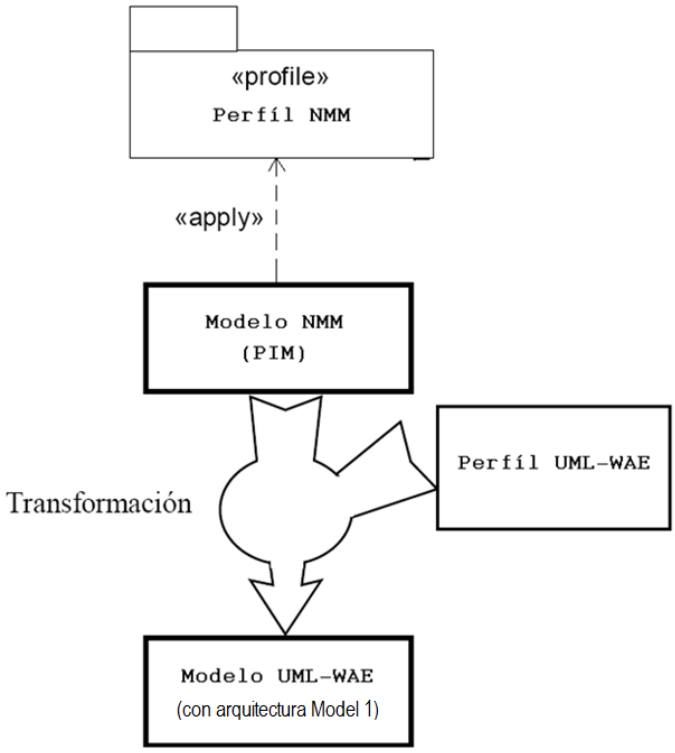
transformados a modelos UML-WAE que expresen una arquitectura concreta. A continuación se mostrará el proceso de transformación que toma un modelo NMM desarrollado utilizando el *Perfil NMMp* discutido en el Capitulo 3 como entrada y produce su correspondiente modelo UML-WAE con *Perfil WAE* también discutido en el Capitulo 3 con arquitectura Model 1. Para realizar las transformaciones se utiliza el lenguaje QVT-*Operational mapping*s (OMG-QVT, 2009) implementado en la herramienta CASE Borland Together 2008.

Model Driven Architecture (MDA) (OMG-MDA, 2003) es una iniciativa de OMG (OMG, 2010) que defiende la definición de modelos como elementos de primera clase para el diseño e implementación de un sistema. En MDA, las actividades más importantes pasan a ser las de modelado de los diferentes aspectos de un sistema y la definición de transformaciones de un modelo a otro de forma que puedan ser automatizadas.

Aplicar el enfoque MDA en este trabajo consiste en definir las transformaciones automáticas entre un modelo NMM (modelo independiente de la arquitectura) y un modelo UML-WAE (modelo con una arquitectura concreta), de forma que a partir del modelo NMM del sistema, de la descripción del modelo destino UML-WAE y de una serie de reglas de transformación, se pueda obtener el modelo UML-WAE con la arquitectura seleccionada de la forma más automatizada posible.

La idea es entonces utilizar fundamentalmente los estereotipos y valores etiquetados del Perfil NMMp para “marcar” los elementos de un modelo NMM y producir el correspondiente modelo UML-WAE expresado también en términos del Perfil UML-WAE con la arquitectura seleccionada (arquitectura Model 1 en este caso). La figura 4-2 ilustra este proceso.

Figura 4-2 Representación gráfica de la transformación NMM a UML-WAE



4.2.1 Transformación de los diagramas de página NMM

En (Navarro et al., 2008b) se presenta la tabla 4-1 donde se describe las reglas de transformación entre los elementos de los diagramas de página NMM y los elementos de los diagramas UML-WAE con arquitectura Model 1.

La transformación mostrada en la tabla 4-1 asume la existencia de una clase "*fachada*" (Gamma et al., 2004) que centraliza la lógica de negocio de la aplicación. Esta clase se creará automáticamente en el modelo UML-WAE generado durante el proceso de transformación cuando se determine que el modelo NMM requiere el procesamiento de lógica de negocio en el lado del servidor (es decir, cuando el modelo NMM incluya anclas computacionales).

El flujo de información entre la capa de lógica de negocio y las páginas `«Server Page»` en el modelo UML-WAE de salida, que en la arquitectura Model 1 son las encargadas de recibir la petición, coordinar el procesamiento y enviar la respuesta al cliente, se implementará utilizando objetos de transferencia de datos (DTO) (Fowler, 2003) cuya estructura interna dependerá del modelo de datos de la aplicación.

Cada funcionalidad que la aplicación exponga y que requiera ejecución de lógica de negocio en el lado del servidor (que en los modelos NMM se representan a través de

anclas computacionales) será implementada en los modelos UML-WAE a través de una clase "*Application Service*" (Fowler, 2003) en la capa de lógica de negocio. Cada vez que se defina una nueva clase *Application Service* en el modelo, se agregará automáticamente una dependencia desde la clase fachada a esta nueva clase, de igual forma, cada vez que se defina una nueva clase estereotipada con `«Server Page»` se agregará automáticamente una dependencia desde esta a la clase fachada.

Tabla 4-1 Transformación de diagramas de página NMM en UML-WAE con arquitectura Model 1

Elemento NMM	Elemento UML-WAE con arquitectura Model 1
Página estática *p.*	Página de cliente *p.*
Página dinámicamente creada *p.*	Página de cliente *p* generada por una Página de servidor.
Ancla de recuperación *a*	-
Ancla computacional *a* fuera de un formulario	-
Ancla computacional *a* en un formulario dentro de una Página *p.*	Formulario *a* agregado a una Página de cliente *p.*
Enlace desde Ancla de recuperación *a* dentro de una Página *p1* a una Página estática *p2.*	Enlace desde Página *p1* a Página de cliente *p2.*
Enlace desde Ancla computacional *a* fuera de un formulario dentro de una página *p1* a una Página dinámicamente creada *p2.*	Enlace desde la Página *p1* a la Página de servidor *aSP* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la Página de servidor *aSP* y la clase *Application Service aAS* hacia ellas + dependencia *build* desde la Página de servidor *aSP* hasta Página de cliente *p2.*
Enlace desde Ancla computacional	Dependencia *submit* desde el Formulario *a* a la

a en un formulario dentro de una página *p1* a una Página dinámicamente creada *p2*.	Página de servidor *aSP* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la Página de servidor *aSP* y la clase *Application Service aAS* hacia ellas + dependencia *build* desde la Página de servidor *aSP* hasta Página de cliente *p2*.

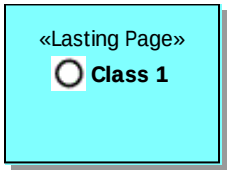
A continuación se mostrará la implementación en Borland Together 2008 de algunas de las reglas de transformación mostradas en la tabla 4-1. Las reglas de transformación completas son mostradas en el apéndice A.

Página estática «Lasting Page»

Una página estática `«Lasting Page»` en NMM representa una página Web que no requiere ser generada a través de procesos computacionales, por lo tanto es directamente transformada a una clase estereotipada `«Client Page»` en UML-WAE. Esta clase representa una página que es administrada por un navegador Web, es decir, solo puede contener algún lenguaje de marcado como HTML ó XML y código Java Script.

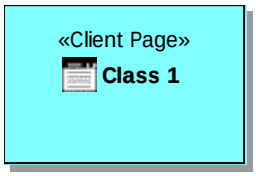
- Elemento fuente en NMM: En la figura 4-3 se muestra la representación visual de una página estática haciendo uso del perfil NMMp desarrollado en el Capitulo 3 de este trabajo.

Figura 4-3 Representación gráfica de página estática en NMM



- Elemento destino en UML-WAE: En la figura 4-4 se muestra la representación visual de una página `«Client Page»` obtenida a partir del modelo NMM de la figura 4-3, haciendo uso del perfil UML-WAE desarrollado en el Capitulo 3 de este trabajo.

Figura 4-4 Representación gráfica de página estática en UML-WAE



- Regla de transformación: la regla de transformación es directa, para cada clase estereotipada con «Lasting Page» en un modelo NMM de entrada se genera una clase estereotipada con «Client Page» en el modelo UML-WAE de salida. Con la clausula "*When*" se asegura que la regla será aplicada solo a clases estereotipadas con «Lasting Page», la nueva clase producida conservará el mismo nombre y descripción.

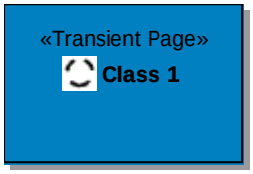
```
mapping uml20::classes::Class::ToClientPage() :
uml20::classes::Class
when
{
            self.stereotypes = OrderedSet{'Lasting Page'}
}
{
      object
      {
            name := self.name;
            description := self.description;
            stereotypes := OrderedSet { 'Client Page' };
      }
}
```

Página dinámicamente creada «Transient Page»

Una página dinámica «Transient Page» en NMM representa una página Web generada por medio de procesos computacionales lanzados como resultado de la interacción con la aplicación Web. Por lo tanto, se corresponde en UML-WAE con una clase estereotipada con «Server Page» que construye ("*Build*") a una clase estereotipada «Client Page». En nuestro modelo, la clase estereotipada con «Server Page» en UML-WAE representa una página Web que interactúa con las capas intermedias de la aplicación y los recursos del lado del servidor.

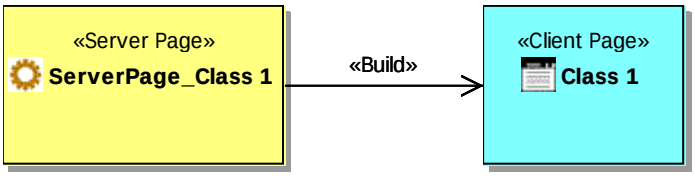
- Elemento fuente en NMM: En la figura 4-5 muestra la representación visual de una página dinámica haciendo uso del perfil NMMp desarrollado en el Capitulo 3 de este trabajo.

Figura 4-5 Representación gráfica de página dinámica en NMM



- Elemento destino en UML-WAE: En la figura 4-6 se muestra la representación visual de una página `«Server Page»`, una página `«Client Page»` y una dependencia `«Build»` obtenidas a partir del modelo NMM de la figura 4-5, haciendo uso del perfil UML-WAE desarrollado en el Capitulo 3 de este trabajo.

Figura 4-6 Representación gráfica de página dinámica en UML-WAE



- Reglas de transformación: en este caso se aplican tres reglas de transformación para cada clase `«Transient Page»` en el modelo NMM de entrada. En la primera, la clase `«Transient Page»` es transformada a la clase `«Client Page»` conservando su mismo nombre y descripción. Con la segunda regla, la misma clase `«Transient Page»` es transformada a la clase `«Server Page»`, aquí por razones de legibilidad se le antepone la palabra "ServerPage" al nombre. Y con la tercera regla se construye la dependencia `«Build»` entre las clases construidas anteriormente.

```
mapping uml20::classes::Class::ToClientPage() : uml20::classes::Class
when
{
            self.stereotypes = OrderedSet{'Transient Page'}
}
{
      object
```

```
        {
                name := self.name;
                description := self.description;
                stereotypes := OrderedSet { 'Client Page' };
        }
}

mapping uml20::classes::Class::ToServerPage() : uml20::classes::Class
when
{
                self.stereotypes = OrderedSet{'Transient Page'}
}
{
        object
        {
                name := 'ServerPage_' + self.name;
                description := self.description;
                stereotypes := OrderedSet { 'Server Page' };
        }
}

mapping uml20::classes::Class::ToBuild() : uml20::classes::Dependency
{
        object
        {
                client      := self.resolve(uml20::classes::Class)->
                               select(c | c.stereotypes->
                               includes('Server Page'));
                supplier    := self.resolve(uml20::classes::Class)->
                               select(c | c.stereotypes->
                                       includes('Client Page'));
                stereotypes := OrderedSet { 'Build' };
        }
}
```

Enlace desde Ancla computacional en un formulario «Form Computing Anchor» dentro de una página a una página dinámicamente creada «Transient Page»

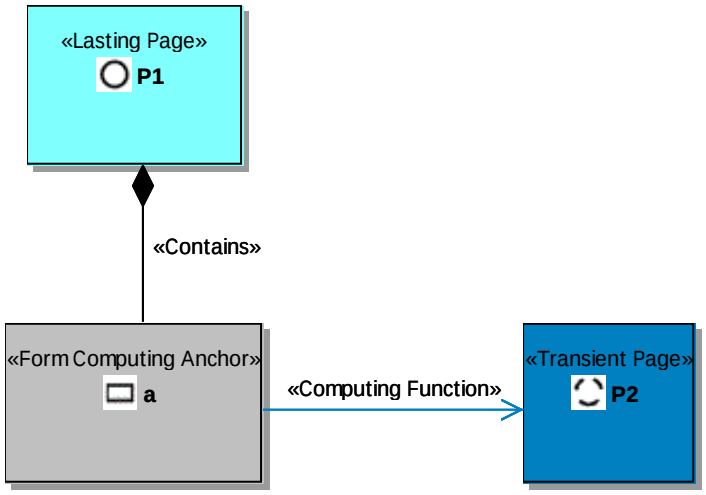
Como se comentó anteriormente, la presencia de anclas computacionales en los modelos NMM de entrada implica crear la capa de lógica de negocio en el modelo UML-WAE con arquitectura Model 1 de salida. La clase «Form Computing Anchor» en primera instancia, debe ser transformada a una clase «Form» que representa un

conjunto de campos de entrada con la capacidad de enviar ("*Submit*") esta información al servidor lanzando un proceso computacional.

Este proceso computacional será ejecutado por una clase `«Application Service»`, de forma que la clase `«Form Computing Anchor»` debe ser transformada también en una clase con este estereotipo. También es necesario agregar un nuevo método en la clase fachada que represente el nuevo proceso computacional, crear las clases `«DTO Input»` y `«DTO Output»` para modelar el flujo de información entre la capa de negocio y la clase `«Server Page»`, y por último generar las relaciones de dependencia adecuadamente.

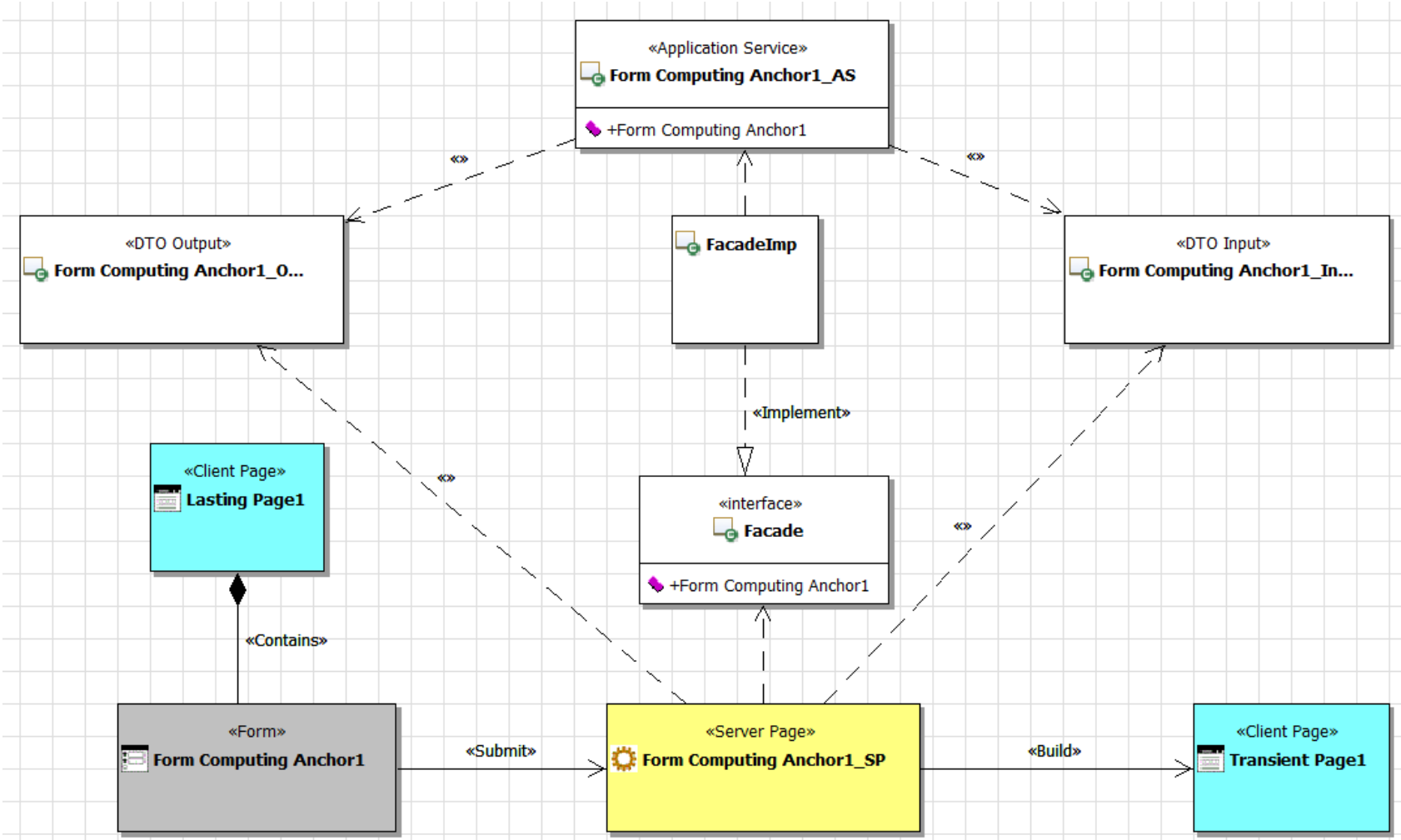
- Elemento fuente en NMM: En la figura 4-7 se muestra la representación visual de una página estática, una página dinámica, una ancla computacional y sus relaciones de dependencia haciendo uso del perfil NMMp desarrollado en el Capitulo 3 de este trabajo.

Figura 4-7 Representación gráfica de ancla computacional en NMM



- Elemento destino en UML-WAE: En la figura 4-8 se muestra la representación visual de un modelo UML-WAE con arquitectura Model 1 generado a parir del modelo NMM de la figura 4-7, haciendo uso del perfil UML-WAE desarrollado en el Capitulo 3 de este trabajo.

Figura 4-8 Representación gráfica de ancla computacional en UML-WAE con arquitectura Model 1



- Reglas de transformación: a continuación se muestra cuatro reglas de transformación para este caso. En la primera, la clase «Form Computing Anchor» es transformada a la clase UML-WAE «Form» conservando el nombre y la descripción. Con la segunda regla a partir de la clase «Form Computing Anchor» se crean las clases «Application Service», «DTO Input» y «DTO Output» de la capa de negocio de la arquitectura Model 1. Con la tercera regla se generan las dependencias adecuadas entre las clases de la capa de lógica negocio y la clase «Server Page». Por último, a través la cuarta regla se genera el método que representa la nueva funcionalidad que será agregado a la clase fachada.

```
mapping uml20::classes::Class::ToForm() : uml20::classes::Class
when
{
            self.stereotypes = OrderedSet{'Form Computing Anchor'}
}
{
      object
      {
            name := self.name;
            description := self.description;
            stereotypes := OrderedSet { 'Form' };
      }
}

query uml20::classes::Class::CreateM1BussinessLayerClasses() :
Set(uml20::classes::Class)
{
      Set
      {
            self.ToApplicationService(self.name + '_AS',
                        'Application Service'),
            self.ToDTOClass(self.name + '_InputTransfer',
                  'DTO Input'),
            self.ToDTOClass(self.name + '_OutputTransfer',
                  'DTO Output')
      }
}

query uml20::classes::Class::CreateM1BussinessLayerDependencies() :
Set(uml20::classes::Dependency)
```

```
{
        Set
        {
                self.CreateDependency(self, 'Server Page',
                                      'DTO Input'),
                self.CreateDependency(self, 'Server Page',
                                      'DTO Output'),
                self.CreateDependency(self, 'Application Service',
                                      'DTO Input'),
                self.CreateDependency(self, 'Application Service',
                                      'DTO Output'),
                self.CreateDependency(self, 'Server Page',
                                      'Facade'),
                self.CreateDependency(self, 'FacadeImp',
                                      'Application Service')
        }
        }

mapping AddOperationInClass(in FormComputingAnchor :
uml20::classes::Class) : uml20::kernel::features::Operation
{
        object
        {
                name := FormComputingAnchor.name;
                visibility := uml::kernel::VisibilityKind::PUBLIC;
        }
}
```

4.2.2 Transformación de los diagramas de región NMM

Con respecto a la interfaz de usuario, UML-WAE utiliza clases estereotipadas «frameset» y «target». Una clase con estereotipo «frameset» representa un objeto contenedor que mapea directamente a elementos HTML etiquetados con <frameset>. Una clase con estereotipo «target» representa una ventana con nombre o una instancia del navegador que puede ser referenciada por una página Web del lado del cliente y debe estar contenida en una o varias clases «frameset».

Los elementos NMM estereotipados con «windows» son transformados en clases UML-WAE estereotipadas con «frameset» y los elementos NMM estereotipados con «region» son transformados en clases UML-WAE estereotipadas con «target». La relación NMM entre los elementos «windows» y «region» es

transformada en una relación de agregación en los modelos UML-WAE. Esta transformación es valida independientemente de la arquitectura.

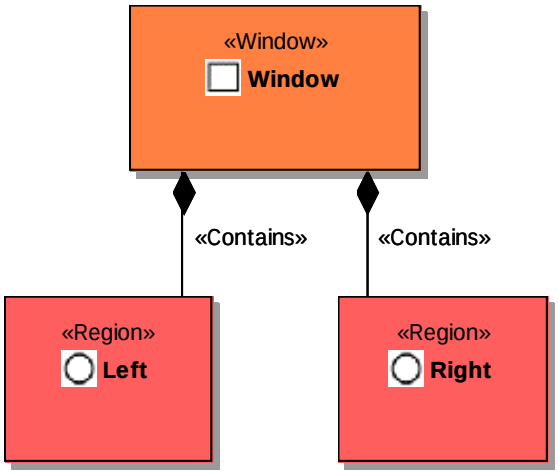
En (Navarro et al., 2008b) se presenta la tabla 2-2 donde se describe las reglas de transformación entre los elementos de los diagramas de región NMM y los elementos de los diagramas UML-WAE.

Tabla 4-2 Transformación de diagramas de región NMM en UML-WAE

Elemento NMM	Elemento UML-WAE con arquitectura Model 1
Ventana *w.*	*Frameset w.*
Región *r.*	*Target r.*
Conexión desde ventana *w* a región *r.*	Agregación desde *Frameset w a Target r.*

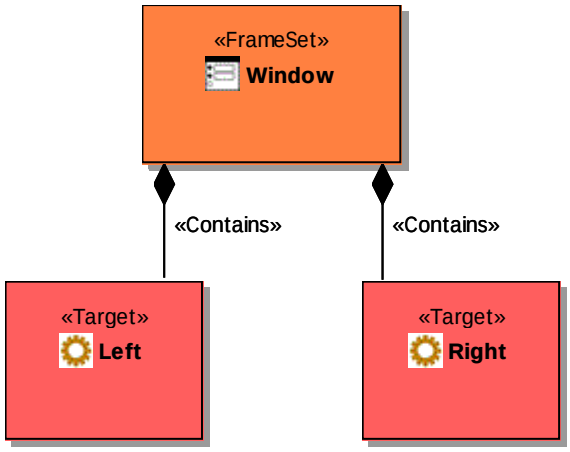
- Elemento fuente en NMM: En la figura 4-9 se muestra la representación visual de un diagrama de regiones haciendo uso del perfil NMMp desarrollado en el Capitulo 3 de este trabajo.

Figura 4-9 Representación gráfica de diagramas de región en NMM



- Elemento destino en UML-WAE: en la figura 4-10 se muestra la representación visual del modelo UML-WAE generado a parir del modelo NMM de la figura 4-9, haciendo uso del perfil UML-WAE desarrollado en el Capitulo 3 de este trabajo.

Figura 4-10 Representación gráfica de diagramas de región en UML-WAE



4.2.3 Transformación de los diagramas de mezcla NMM

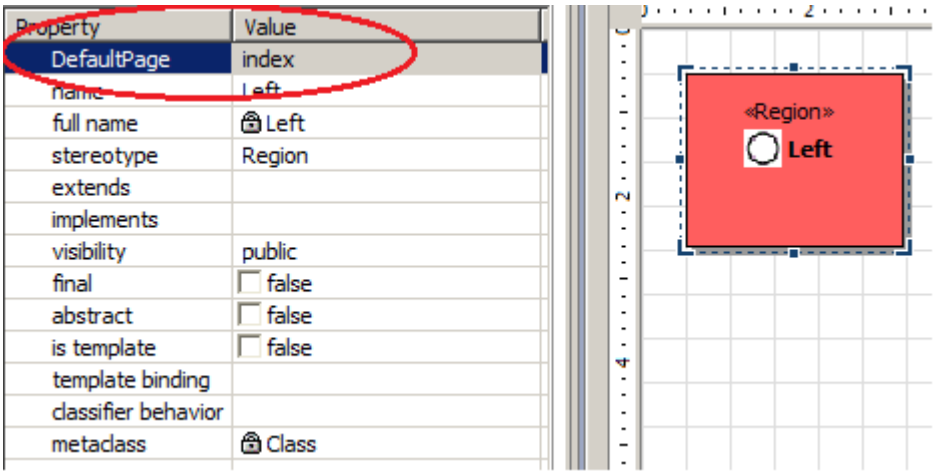
Como se comentó en el Capitulo 3, la función de asignación de página por defecto a los elementos `«Region»` en NMM se ha implementado como valor etiquetado en el perfil NMMp, por lo tanto cada elemento `«Region»` de un modelo NMM posee un atributo llamado "DefaultPage" que contiene la página por defecto asignada a cada región.

De igual forma se ha definido en el perfil UML-WAE el valor etiquetado "DefaultPage" asociado a las clases estereotipadas con `«Target»`, de forma que todos los elementos `«Target»` de un modelo UML-WAE posee un atributo llamado "DefaultPage" que contiene la página por defecto asignada a cada clase estereotipada `«Target»`.

Por lo tanto la transformación de la función de asignación de página por defecto a los elementos `«Region»` en NMM se hace simplemente copiando el valor contenido en el atributo "DefaultPage" de los elementos `«Region»` de NMM al atributo "DefaultPage" de los elementos `«Target»` de UML-WAE.

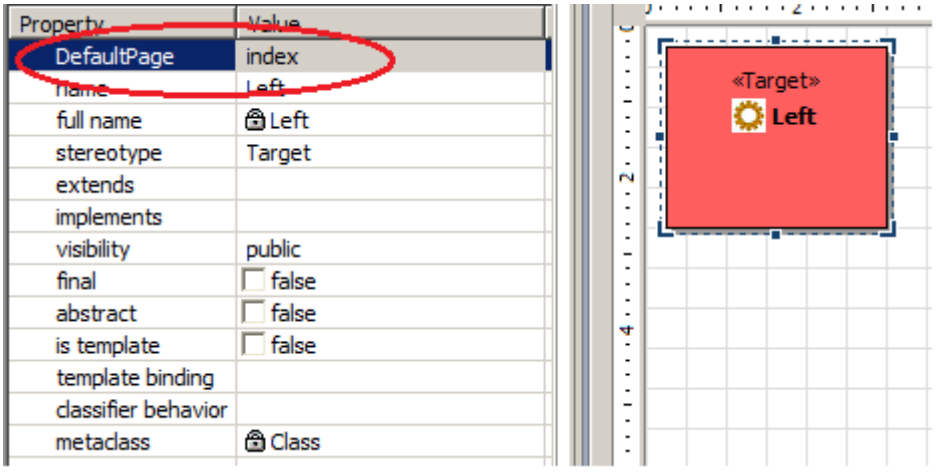
- Elemento fuente en NMM: en la figura 4-11 se muestra la representación visual de la función de asignación de página por defecto a las regiones NMM haciendo uso del perfil NMMp desarrollado en el Capitulo 3 de este trabajo.

Figura 4-11 Representación gráfica de la función de asignación de página por defecto a regiones en NMM



- Elemento destino en UML-WAE: en la figura 4-12 se muestra la representación visual del modelo UML-WAE generado a partir del modelo NMM de la figura 4-11, haciendo uso del perfil UML-WAE desarrollado en el capitulo 3 de este trabajo.

Figura 4-12 Representación gráfica de la función de asignación de página por defecto a regiones en UML-WAE



- Reglas de transformación:

```
mapping uml20::classes::Class::ToTarget(inout Target :
uml20::classes::Class) : uml20::classes::Class
when
{
```

```
                self.stereotypes = OrderedSet{'Region'}
}
{
        init
        {
                Target.name := self.name;
                Target.description := self.description;
                Target.stereotypes := OrderedSet { 'Target' };
                Target.setPropertyValue('DefaultPage',
                        self.getPropertyValue('DefaultPage'));
                result := Target;
        }
}
```

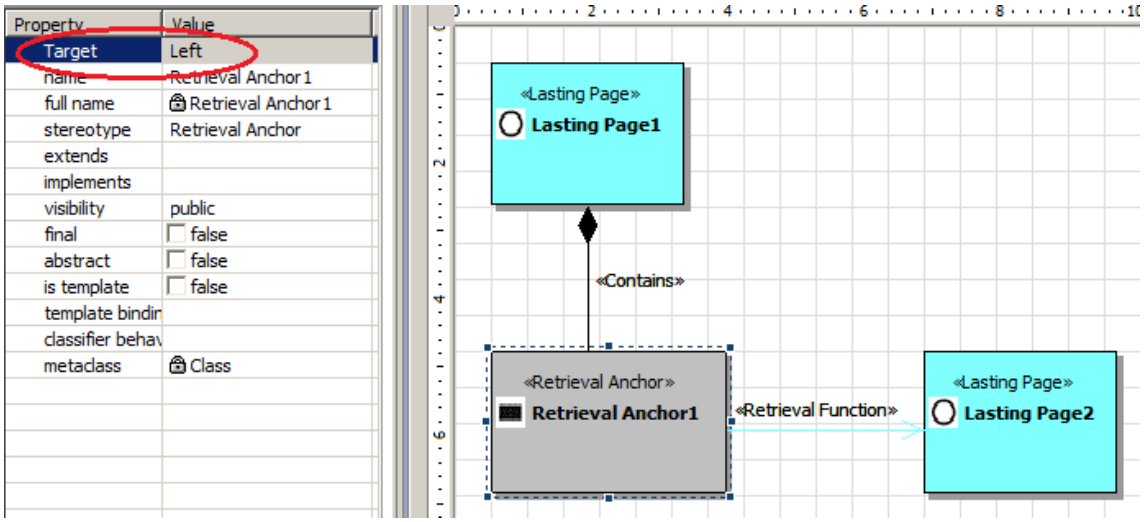
De forma semejante, la función de asignación de la región destino a las anclas NMM se ha implementado asociando un valor etiquetado llamado “Target” a los elementos «Retrieval Anchor», «Form Computing Anchor» y «Non Form Computing Anchor» en el perfil NMMp. Por lo tanto cada uno de estos elementos en un modelo NMM posee un atributo llamado “Target” que contiene la región por defecto asignada a cada ancla.

De igual forma, en el perfil UML-WAE se ha asociado el valor etiquetado “Target” a los elementos «Link» y «Submit», de forma que estos elementos posean el atributo “Target” en los modelos UML-WAE.

La transformación de la función de asignación de la región destino de las anclas NMM se efectúa copiando el valor del atributo “Target” de las anclas NMM a los elementos «Link» ó «Submit» de los modelos UML-WAE.

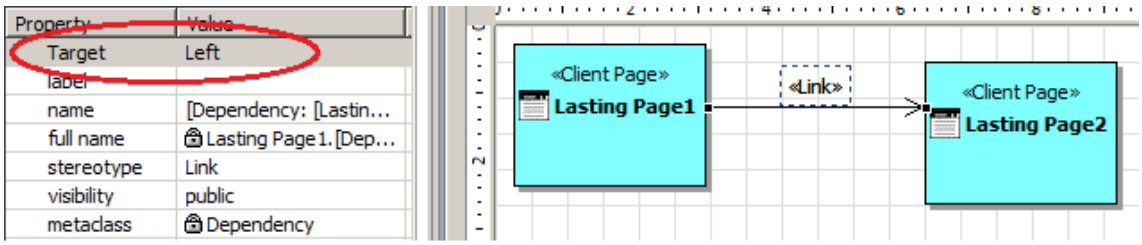
- Elemento fuente en NMM: en la figura 4-13 se muestra la representación visual de la función de asignación de la región destino a las anclas NMM haciendo uso del perfil NMMp desarrollado en el capitulo 3 de este trabajo.

Figura 4-13 Representación gráfica de la función de asignación de región destino a las anclas en NMM



- Elemento destino en UML-WAE: en la figura 4-14 se muestra la representación visual del modelo UML-WAE generado a partir del modelo NMM de la figura 4-13, haciendo uso del perfil UML-WAE desarrollado en el capitulo 3 de este trabajo.

Figura 4-14 Representación gráfica de la función de asignación de región destino a las anclas en UML-WAE



- Reglas de transformación:

```
mapping uml20::classes::Class::ToLink(inout Link :
        uml20::classes::Dependency) : uml20::classes::Dependency
{
                init
                {
                        Link.client := self.dependencies->
                                       select(d | d.stereotypes->
                                       includes('Contains'))->
                                       collect(dc |
```

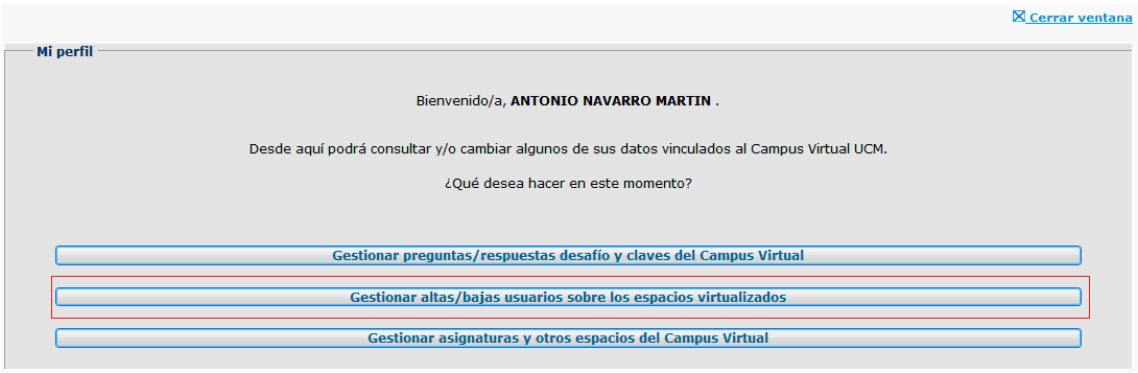
```
                dc.client.oclAsType(uml20::classes::Class))

                    Link.supplier := self.dependencies->select(d |
                    d.stereotypes->includes('Retrieval
                    Function'))->collect(dc |
                    c.supplier.oclAsType(uml20::classes::Class));

                    Link.stereotypes := OrderedSet{ 'Link' };

                    Link.setPropertyValue('Target',
                              self.getPropertyValue('Target'));
            }
    }
```

4.3 Ejemplo

En las secciones anteriores se ha introducido y definido las reglas de transformación QVT para convertir modelos NMMp en modelos UML-WAE con arquitectura Model 1. En esta sección se mostrará un ejemplo real de su aplicación, sin embargo, para facilitar la ilustración del proceso se comenzará comentando una funcionalidad de una aplicación Web desplegada en el entorno de producción del *Campus Virtual* de la *Universidad Complutense de Madrid (UCM)*, después se modelará la funcionalidad haciendo uso del perfil NMMp y por último, a partir de este se obtendrá el modelo UML-WAE con arquitectura Model 1.

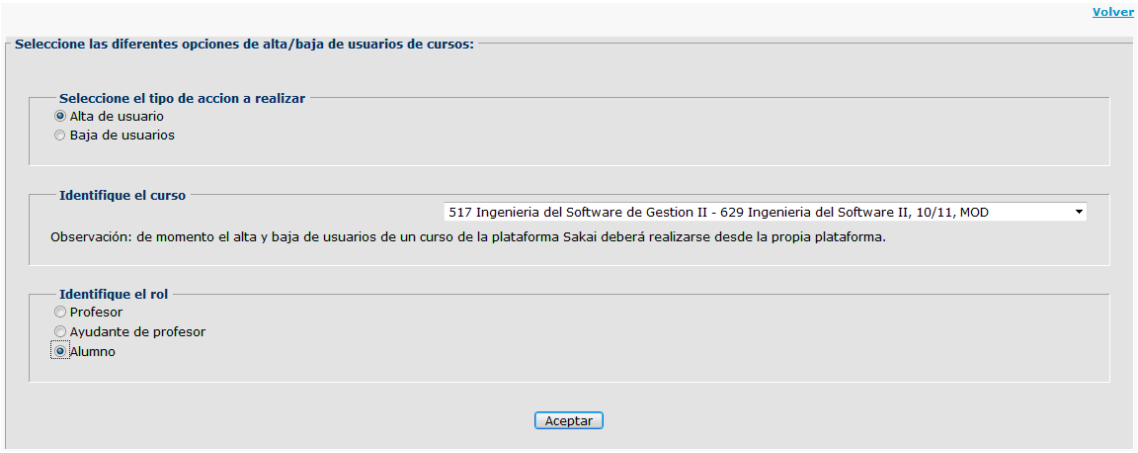
El Campus Virtual de la UCM ofrece la funcionalidad de inscripción de usuarios en diferentes cursos a través de la opción “Gestionar altas/bajas usuarios sobre los espacios virtualizados”, como se muestra en la figura 4-15.

Figura 4-15 Página Web que ofrece la funcionalidad de inscripción de usuarios en diferentes cursos



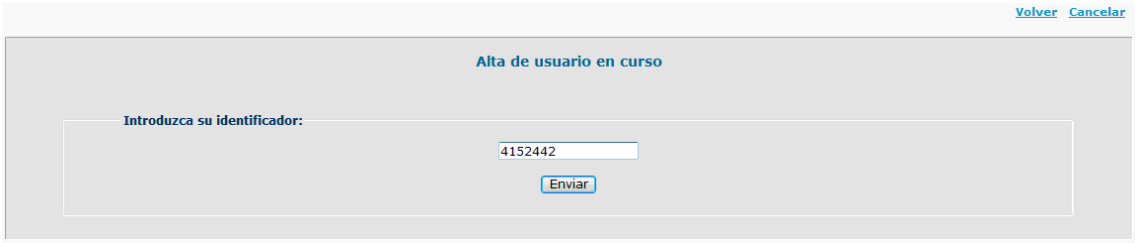
Cuando un usuario accede a la funcionalidad a través de esta opción, se muestra en su navegador una nueva página Web con un formulario donde se le piden los datos para llevar a cabo la inscripción, como se muestra en la figura 4-16. Notar que aunque esta página ofrece también la opción de dar de baja a un usuario en un curso, esta funcionalidad no es considerada aquí.

Figura 4-16 Página Web que recolecta los datos para llevar a cabo una inscripción.



Una vez el usuario selecciona las opciones para dar de alta la inscripción y pulsa el botón “Aceptar”, si las opciones seleccionadas son correctas, la aplicación le presenta la página Web de la figura 4-17, donde a través de un formulario se le pide su identificación. En caso contrario, la aplicación le volverá a mostrar la misma página de la figura 4-16 manteniendo las opciones seleccionadas y con un mensaje de error.

Figura 4-17 Página Web que recolecta el identificador de usuario para llevar a cabo una inscripción.



Después de introducir el identificador de usuario en la página de la figura 4-17 y si los datos son correctos, la aplicación le presenta la página Web de la figura 4-18 donde se le informa que el proceso ha sido satisfactorio. En caso contrario, la aplicación le

volverá a mostrar la misma página de la figura 4-17 manteniendo el dato introducido y con un mensaje de error.

Figura 4-18 Página Web que informa que el proceso de inscripción ha sido satisfactorio.

En la figura 4-19 se muestra el modelo NMMp que caracteriza la estructura navegacional de la funcionalidad anterior. Como puede observarse, la página `management` incluye un ancla computacional fuera de formulario llamada `toUserEnrollment`, la cual da acceso a la página `userEnrollment`. Esta página es la encargada de recolectar la información para la inscripción del usuario usando un ancla computacional en un formulario llamada `toGetUserId`, la cual invoca los componentes responsables de validar los datos. Esta validación puede ser satisfactoria, proporcionando acceso a la página `getUserId`, ó puede ser incorrecta, redirigiendo el flujo navegacional a la página `userEnrollment`, obligando a introducir los datos correctamente. Esta redirección es descrita en el modelo NMMp enlazando el ancla `toGetUserId` con el ancla `toUserEnrollment`, la cual genera la página `userEnrollment`.

La página `getUserId` recoge el identificador del usuario a inscribir usando el ancla computacional en un formulario llamada `enrollUser`, la cual invoca los componentes responsables de la inscripción de los usuarios. El proceso de la inscripción puede ser satisfactorio, proporcionando acceso a la página `enrollmentOK`, ó puede fallar, redirigiendo el flujo navegacional a la página `getUserId`, obligando a introducir los datos correctamente. Esta redirección es descrita en el modelo NMMp enlazando el ancla `enrollUser` con el ancla `toGetUserId`, la cual genera la página `getUserId`.

Notar que, en teoría, el ancla `toGetUserId` podría redirigir el control a la página `management`, pero teniendo en cuenta los datos que serán proporcionados en tiempo de ejecución, lo más apropiado parece ser redirigir el control a la página `getUserId`. Por lo tanto, este modelo previene a los desarrolladores para evitar volver el control a la página

`management`, controlando los valores de entrada asociados a los procesos invocados por las anclas computacionales.

Finalmente, como se muestra en el diagrama de la figura 4-19, los valores de entrada que el usuario introduce a través de los formularios son especificados usando el atributo `Input` de la dependencia estereotipada con `«Computing Function»`. El diagrama de páginas NMMp mostrado en la figura 4-19 es un diagrama abstracto, independiente de cualquier arquitectura de software y lenguaje de programación, que muestra las páginas y su relación navegacional en la capa de presentación de una aplicación Web.

Cuando las transformaciones definidas en la sección anterior son aplicadas al modelo mostrado en la figura 4-19, el modelo UML-WAE con arquitectura Model 1 mostrado en la figura 4-20 es obtenido.

Figura 4-19 Diagrama de páginas NMMp que modela la funcionalidad de inscripción de usuarios a través de Campus Virtual de la UCM

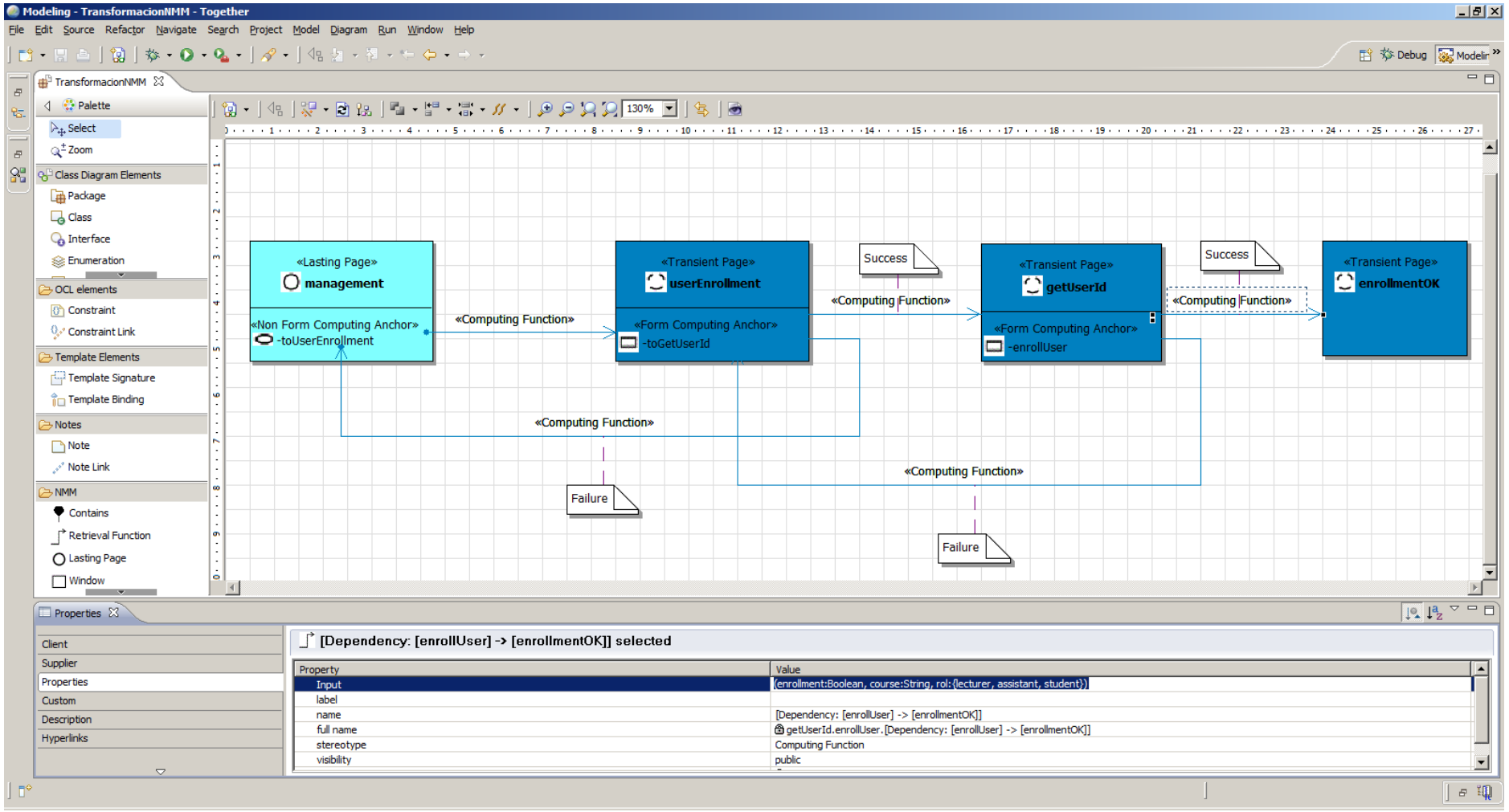
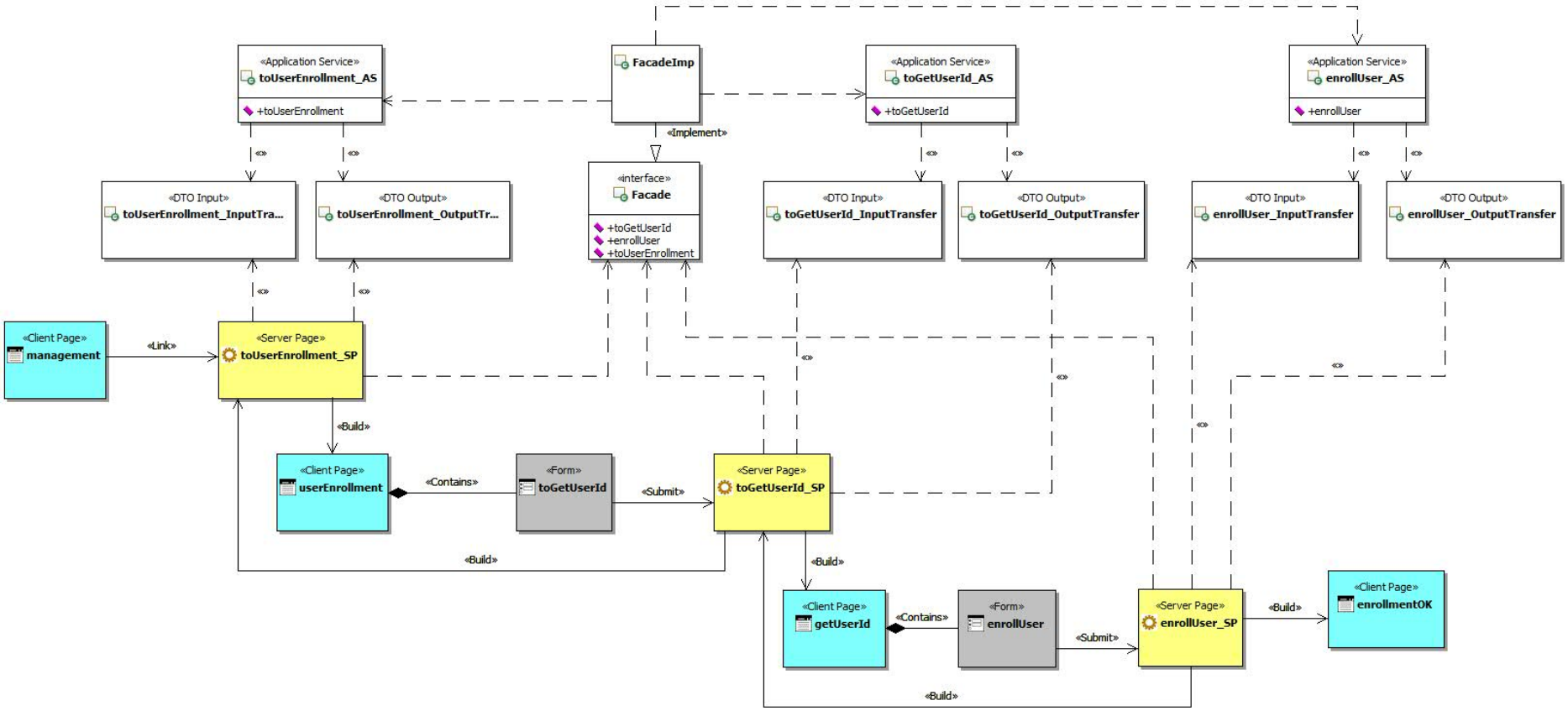


Figura 4-20 Diagrama UML-WAE con arquitectura Model 1 generado automáticamente a partir del diagrama NMMp de la Figura 4-19 que modela la funcionalidad de inscripción de usuarios a través de Campus Virtual de la UCM



En este ejemplo de transformación, las páginas NMMp han sido convertidas en páginas UML-WAE, las anclas computacionales han sido utilizadas para crear todos los componentes computacionales de la capa de negocio. De esta manera, las anclas computacionales NMMp ocultan las acciones invocadas por las páginas de servidor, la fachada responsable de la implementación de las funcionalidades (usando clases especializadas estereotipadas con `«Application Service»`) y los objetos de transferencia de datos que mueven los datos entre la capa de negocio y la capa de presentación. Por ejemplo, la página `getUserId` en la figura 4-19 ha sido transformada en la página de servidor `toGetUserId_SP` y en la página de cliente `getUserId`; mientras que el ancla computacional `enrollUser` ha sido transformada en las clases `enrollUser_InputTransfer`, `enrollUser_OutputTransfer`, `enrollUser_AS` y en el método `enrollUser` de la clase `Facade`. Los enlaces entre anclas y páginas en el diagrama NMMp han sido transformados en enlaces entre los elementos UML-WAE.

Como se muestra en la figura 4-20, UML-WAE se aplica solo a la capa de presentación de las aplicaciones Web, sin embargo el flujo navegacional mostrado claramente por el diagrama NMMp de la figura 4-19 se vuelve confuso. En general los diagramas NMMp son valiosos tanto para usuarios como para desarrolladores, mientras que los diagramas UML-WAE son valiosos solo para los desarrolladores, quienes solo los responsables de desarrollar la estructura navegacional y los artefactos computacionales necesarios para construir la aplicación Web. Es importante señalar que mientras el diagrama NMMp de la figura 4-19 tiene solo siete clases UML (cuatro páginas y tres anclas computacionales), el diagrama UML-WAE de la figura 4-20 tiene veinte clases. Por lo tanto el poder semántico de los elementos NMMp queda demostrado.

Como hemos comentado, el diagrama UML-WAE de la figura 4-20 describe la capa de presentación de un aplicación Web; dado que este diagrama y el diagrama NMMp de la figura 4-19 son diagramas UML, estos pueden ser integrados con el resto de los diagramas UML desarrollados con cualquier herramienta CASE de propósito general. Como puede verse, algunos patrones multicapas son usados en la transformación (*Application Controllers, Transfers Data Objects, Applications Services*) y otros patrones multicapas y SOA pueden ser usados para describir el resto de las capas de la aplicación.

De esta forma, el enfoque NMMp estimula el uso de patrones arquitectónicos para el diseño de aplicaciones Web, como el sector industrial lo hace, pero, usando al mismo tiempo mapas navegacionales para incrementar el nivel de abstracción del diseño, como el sector académico lo promueve. De esta forma, un equilibrio entre el enfoque basado en patrones y el enfoque basado en notaciones es logrado en NMMp.

5. De NMM a UML-WAE Model 2

Esta sección define las transformaciones QVT Operational Mappings para traducir PIMs independientes de la arquitectura NMMp en PIMs UML WAE con una arquitectura de presentación Model 2.

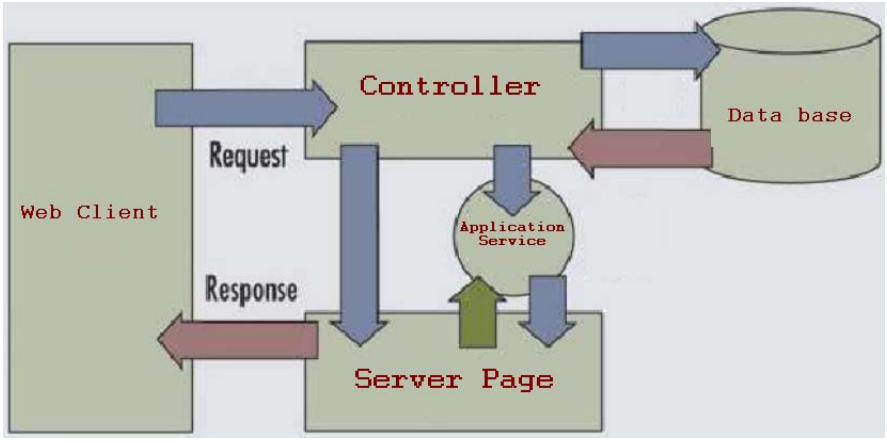
5.1 Arquitectura Model 2

Model 2 es un diseño arquitectónico para aplicaciones Web diseñado para resolver las deficiencias encontradas en la arquitectura Model 1. La arquitectura Model 2 es una implementación del lado del servidor del patrón de diseño Model-View-Controller (MVC). Este patrón hace cumplir la separación entre la forma en que el modelo de la aplicación es modelado (*Model*) y la forma en la que es presentado (*View*), esto requiere un componente separado para coordinar la relación entre ellos ("*Controller*").

La arquitectura Model 2 utiliza un "*Controller*" para manejar todas las peticiones de procesamiento y delegar cada petición a una página Web del lado del servidor ("*Server Page*") diferente para componer la presentación y responder a la petición, a diferencia de la arquitectura Model 1 en la cual el controlador y la vista coexisten dentro del mismo componente.

Como se muestra en la figura 5-1, todas las peticiones hechas a la aplicación Web son recibidas por un controlador, dependiendo del tipo de petición, el controlador es responsable de interactuar con los servicios de la aplicación apropiados y redirigir la petición a una página "*Server Page*", la cual construye la respuesta adecuada basada en los resultados de la ejecución de los servicios de la aplicación.

Figura 5-1 Arquitectura Model 2 (*Model-View-Controller*)



Existen otras variantes de esta arquitectura, tales como proporcionar múltiples controladores para distribuir la funcionalidad de la administración de peticiones. Es importante resaltar que independientemente de las variantes, en la arquitectura Model 2 los componentes siempre deben tener un rol bien definido. Algunas de las debilidades de la arquitectura Model 1 que son superadas con la arquitectura Model 2 son:

- Mantenibilidad: los problemas de mantenibilidad asociados con la arquitectura Model 1 son resultados directos de la unificación del controlador y la vista en un mismo componente. Ya que el procesamiento de la lógica de negocio y la generación del contenido se ejecutaban en la misma "*Server Page*", ésta resultaba confusa y por la tanto difícil de mantener. Separando la lógica de la aplicación de la presentación usando componentes MVC se facilita el desarrollo de código especializado para cada una de estas tareas resultando en una aplicación más flexible y mantenible.
- Seguridad: como existe un único componente controlador como punto de acceso para todas las peticiones, es un lugar excelente para ubicar alguna forma de autenticación. Cuando un usuario hace una petición que debe pasar algún mecanismo de autenticación, el controlador puede continuar con la petición de forma usual, si la autenticación ha sido satisfactoria, ó, alternativamente redirigir la petición a una página de error apropiada (por ejemplo a una página de *login*) donde el error puede ser tratado. Dado que el mecanismo de autenticación existe en un único lugar, cualquier cambio en él debe ser hecho una sola vez, además, es fácil tomar ventaja de los mecanismos de seguridad declarativos.
- Extensibilidad: uno de los mayores beneficios de la arquitectura Model 2 es que todo el procesamiento de la lógica de negocio esta centralizado. Esta centralización proporciona una solución basada en componentes que pueden ser reutilizados ó extendidos reduciendo la posibilidad de que haciendo un cambio cause un efecto no deseado en otros componentes dependientes.

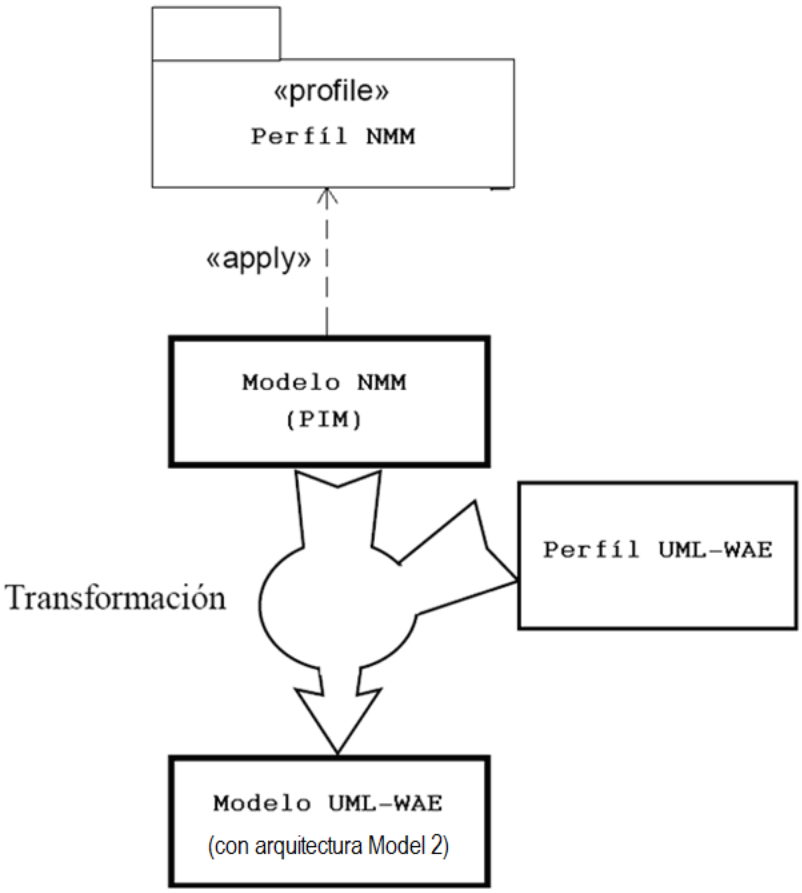
5.2 Transformación

A continuación se mostrará el proceso de transformación que toma un modelo NMM desarrollado utilizando el *Perfil NMMp* discutido en el Capitulo 3 como entrada y produce su correspondiente modelo UML-WAE con *Perfil WAE* también discutido en el Capitulo 3 con arquitectura Model 2. Para realizar las transformaciones se utiliza el

lenguaje QVT-*Operational mapping*s (OMG-QVT, 2009) implementado en la herramienta CASE Borland Together 2008.

El proceso consiste fundamentalmente en utilizar los estereotipos y valores etiquetados del Perfil NMMp para "marcar" los elementos de un modelo NMM y producir el correspondiente modelo UML-WAE expresado también en términos del Perfil UML-WAE con la arquitectura Model 2. La figura 5-2 ilustra este proceso.

Figura 5-2 Representación gráfica de la transformación NMM a UML-WAE



5.2.1 Transformación de los diagramas de página NMM

En (Navarro et al., 2008b) se presenta la tabla 5-1 donde se describe las reglas de transformación entre los elementos de los diagramas de página NMM y los elementos de los diagramas UML-WAE con arquitectura Model 2.

La transformación mostrada en la tabla 5-1 asume la existencia de una clase "*Controller*" que actúa como delegado de negocio para administrar todas las peticiones de procesamiento, una clase "*fachada*" que centraliza la lógica de negocio de la

aplicación y una interfaz "*action*" que define un método llamado "*execute*" para desacoplar el modelo de la aplicación de la vista. La clase "*fachada*" y las clases "*Action*" que implementan la interfaz "*action*" se crearan automáticamente en el modelo UML-WAE generado durante el proceso de transformación cuando se determine que el modelo NMM requiere el procesamiento de lógica de negocio en el lado del servidor (es decir, cuando el modelo NMM incluya anclas computacionales).

El flujo de información de entrada desde la clase "*Controller*" a la capa de lógica de negocio, y de salida desde la capa de lógica de negocio a las páginas `«Server Page»` se implementará utilizando objetos de transferencia de datos (DTO) (Fowler, 2003) cuya estructura interna dependerá del modelo de datos de la aplicación.

Cada funcionalidad que la aplicación exponga y que requiera ejecución de lógica de negocio en el lado del servidor (que en los modelos NMM se representan a través de anclas computacionales) será implementada en los modelos UML-WAE a través de una clase "*Action*" que debe implementar la interfaz "*action*" y de una clase "*Application Service*" (Fowler, 2003) en la capa de lógica de negocio. Cada vez que se defina una nueva clase *Action* en el modelo, se agregará automáticamente una dependencia desde esta a la clase *fachada.* De igual forma, cada vez que se defina una nueva clase *Application Service* en el modelo, se agregará automáticamente una dependencia desde la clase fachada a esta nueva clase.

Tabla 5-1 Transformación de diagramas de página NMM en UML-WAE con arquitectura Model 2

Elemento NMM	Elemento UML-WAE con arquitectura Model 2
Página estática *p.*	Página de cliente *p.*
Página dinámicamente creada *p.*	Página de cliente *p* generada por una Página de servidor.
Ancla de recuperación *a*	-
Ancla computacional *a* fuera de un formulario	-
Ancla computacional *a* en un formulario dentro de una Página *p.*	Formulario *a* agregado a una Página de cliente *p.*

Enlace desde Ancla de recuperación *a* dentro de una Página *p1* a una Página estática *p2.*	Enlace desde Página *p1* a clase *Controller* + dependencia *forward* desde la clase *Controller* a Página de cliente *p2.*
Enlace desde Ancla computacional *a* fuera de un formulario dentro de una página *p1* a una Página dinámicamente creada *p2.*	Enlace desde la Página *p1* a clase *Controller* + clase «Action» *aAction* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la clase *aAction* y la clase *Application Service aAS* hacia ellas + dependencia *forward* desde la clase *Controller* a la página de servidor *aView* + dependencia desde la página de servidor *aView* a la clase de tranferencia de datos *aOutputTransfer* + dependencia *build* desde la Página de servidor *aView* hasta Página de cliente *p2.*
Enlace desde Ancla computacional *a* en un formulario dentro de una página *p1* a una Página dinámicamente creada *p2.*	Dependencia *submit* desde el Formulario *a* a clase *Controller* + clase «Action» *aAction* + operación *a* en la clase Fachada + clase *Application Service aAS* + clases de transferencias de datos *aInputTransfer* y *aOutputTransfer* con dependencias desde la clase *aAction* y la clase *Application Service aAS* hacia ellas + dependencia *forward* desde la clase *Controller* a la página de servidor *aView* + dependencia desde la página de servidor *aView* a la clase de tranferencia de datos *aOutputTransfer* + dependencia *build* desde la Página de servidor *aView* hasta Página de cliente *p2.*

A continuación se mostrará la implementación en Borland Together 2008 de algunas de las reglas de transformación de los *diagramas de página NMM* mostradas en la tabla 5-1; como puede verse, algunas de las reglas son idénticas a las detalladas en el Capitulo 4, además, dado que las reglas de transformación de los *diagramas de región NMM* y los *diagramas de mezcla NMM* son independientes de la arquitectura de los

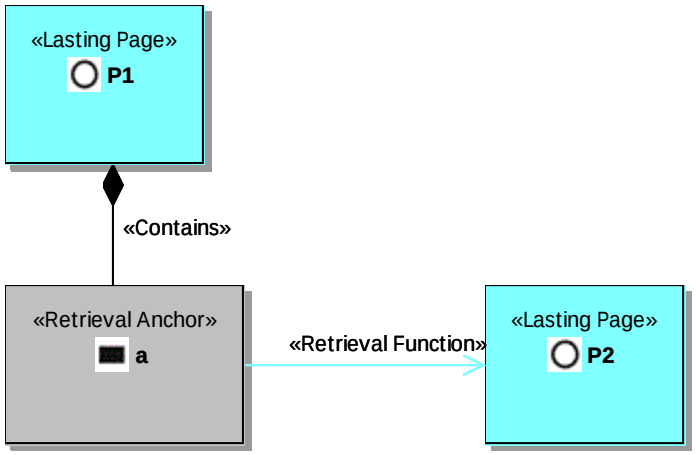
modelos UML-WAE destino, no se comentaran aquí, ya que ya fueron detalladas en el Capitulo 4. Las reglas de transformación completas son mostradas en el apéndice B.

Enlace desde Ancla de recuperación «Retrieval Anchor» dentro de una página a una página estática «Lasting Page»

Como se comentó anteriormente, la arquitectura Model 2 requiere de una clase "*Controller*" para administrar todas las peticiones de los clientes Web, por lo tanto, la clase «**Retrieval Anchor**» debe ser transformada en primera instancia a una clase llamada "Controller" del lado del servidor, si esta no existiese, y crear la asociación estereotipada con «Link» desde la página que inicia el proceso a la clase Controller, y la dependencia "Forward" desde la clase "Controller" a la página destino.

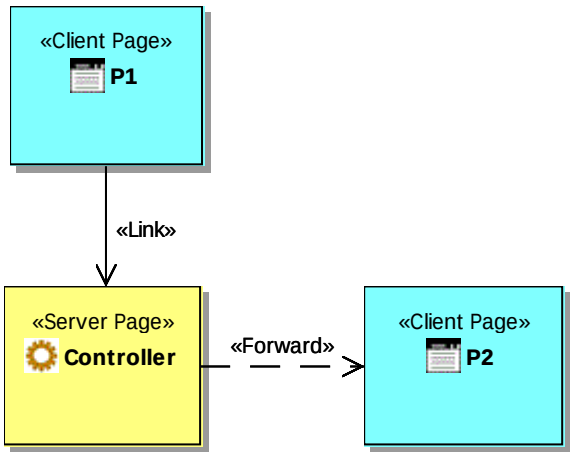
- Elemento fuente en NMM: En la figura 5-3 se muestra la representación visual de una página estática que contiene un ancla de recuperación que puede iniciar un proceso de navegación hasta una segunda página estática haciendo uso del perfil NMMp desarrollado en el Capitulo 3 de este trabajo.

Figura 5-3 Representación gráfica de ancla de recuperación en NMM



- Elemento destino en UML-WAE: En la figura 5-4 se muestra la representación visual de un modelo UML-WAE con arquitectura Model 2 generado a parir del modelo NMM de la figura 5-3, haciendo uso del perfil UML-WAE desarrollado en el Capitulo 3 de este trabajo.

Figura 5-4 Representación gráfica de ancla de recuperación en UML-WAE con arquitectura Model 2



- Reglas de transformación: a continuación se muestra tres reglas de transformación para este caso. En la primera, se comprueba si la clase "`Controller`" ha sido creada previamente, en caso contrario se procede a crearla a través de la segunda regla de transformación. Con la tercera regla de transformación se crea la dependencia `«Forward»`.

```
query uml20::classes::Class::HasBeenConvertedToController() : Boolean
{
	if((self.resolve(uml20::classes::Class)->select(class |
		class.name = 'Controller'))->any() = true) then

		true

	else

		false

	endif
}

mapping uml::together::Model::ToControllerClass() :
uml20::classes::Class
{
	object
	{
		name := 'Controller';
		stereotypes := OrderedSet{ 'Server Page' };
	}
```

```
}

mapping uml20::classes::Dependency::ToForwardDependencies(in NMMModel :
uml::together::Model) : uml20::classes::Dependency
{
        init
        {
                var Supplier :=
                self.supplier.oclAsType(uml20::classes::Class);

                result := NMMModel.CreateDependency(Supplier,
                                                    'Controller',
                                                    'Client Page',
                                                    'Forward');
        }
}
```

Enlace desde Ancla computacional en un formulario «Form Computing Anchor» dentro de una página a una página dinámicamente creada «Transient Page»

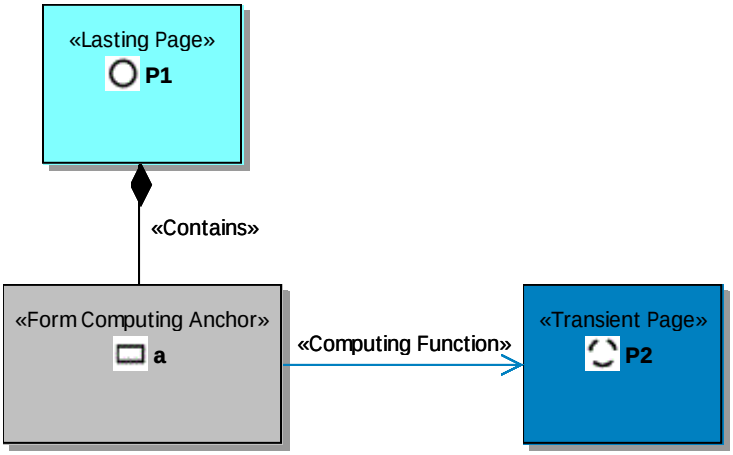
Como se comentó anteriormente, la presencia de anclas computacionales en los modelos NMM de entrada implica crear la capa de lógica de negocio en el modelo UML-WAE con arquitectura Model 2 de salida. La clase «Form Computing Anchor» en primera instancia, debe ser transformada a una clase «Form» que representa un conjunto de campos de entrada con la capacidad de enviar ("*Submit*") esta información al servidor lanzando un proceso computacional. Es necesario también comprobar si la clase "Controller" y la interfaz "action" han sido creadas previamente en la capa de lógica de negocio y en caso negativo proceder a crearlas.

Este proceso computacional será ejecutado por una clase «Application Service» que se desacopla de la clase "Controller" a través de una clase estereotipada con «action» que implementa la interfaz "action", de forma que la clase «Form Computing Anchor» debe ser transformada también en dos clases con estos estereotipos. También es necesario agregar un nuevo método en la clase fachada que represente el nuevo proceso computacional, crear las clases «DTO Input» y «DTO Output» para modelar el flujo de información entre la capa de negocio y la clase «Server Page», y por último generar las relaciones de dependencia adecuadamente.

- Elemento fuente en NMM: En la figura 5-5 se muestra la representación visual de una página estática, una página dinámica, una ancla computacional y sus

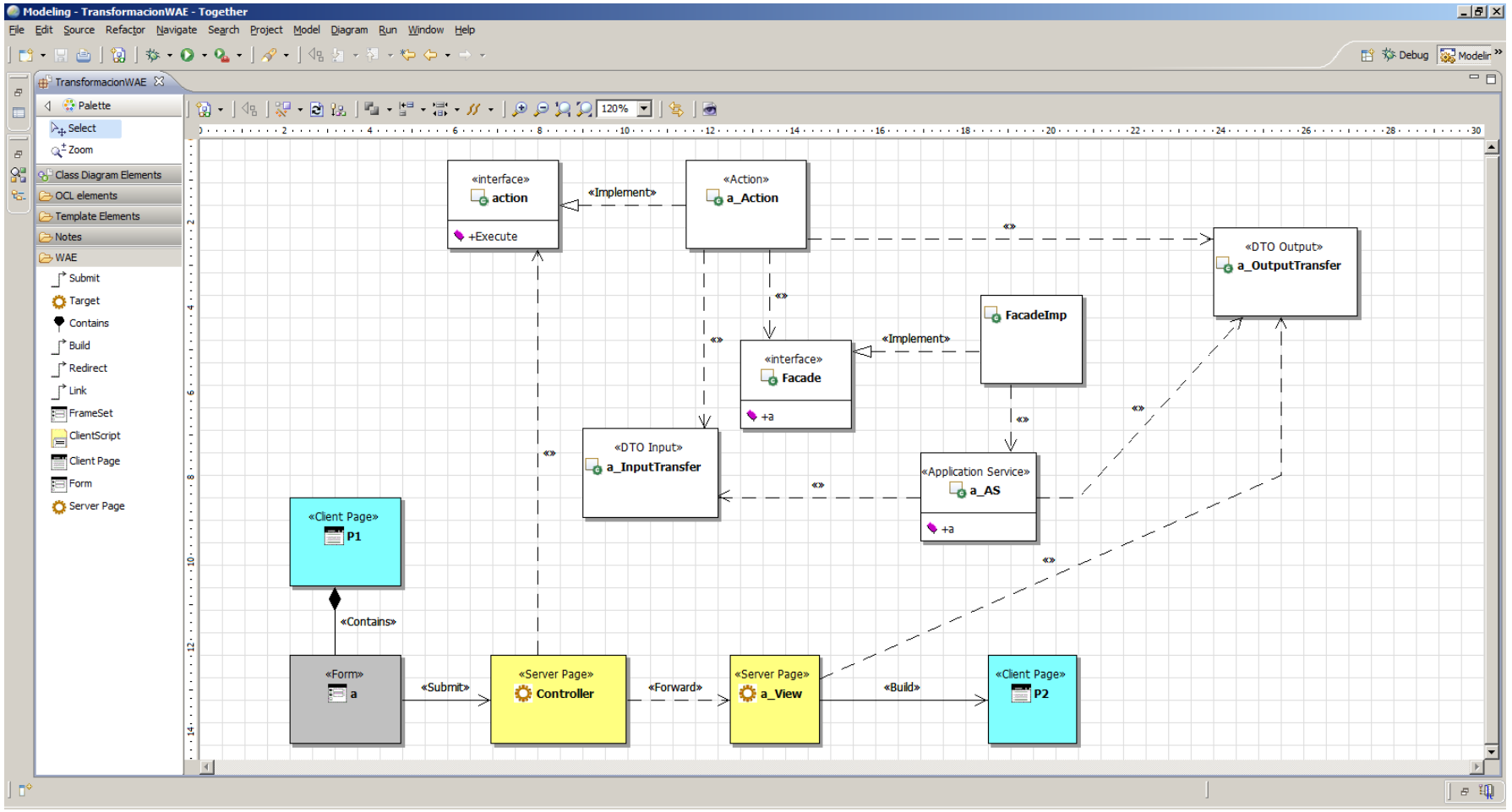
relaciones de dependencia haciendo uso del perfil NMMp desarrollado en el Capitulo 3 de este trabajo.

Figura 5-5 Representación gráfica de ancla computacional en NMM



- Elemento destino en UML-WAE: En la figura 5-6 se muestra la representación visual de un modelo UML-WAE con arquitectura Model 2 generado a parir del modelo NMM de la figura 5-6, haciendo uso del perfil UML-WAE desarrollado en el Capitulo 3 de este trabajo.

Figura 5-6 Representación gráfica de ancla computacional en UML-WAE con arquitectura Model 2



- Reglas de transformación: a continuación se muestra tres reglas de transformación para este caso. En la primera, se crea la interfaz "action" en caso de no haber sido creada antes. Con la segunda regla, a partir de la clase «Form Computing Anchor» se crean las clases «Server Page», «action», «Application Service», «DTO Input» y «DTO Output» de la capa de negocio de la arquitectura Model 2. Y con la tercera regla se generan las dependencias adecuadas entre las clases de la capa de lógica negocio.

```
mapping uml::together::Model::ToActionClass() : uml20::classes::Class
{
       object
       {
          name := 'action';
          stereotypes := OrderedSet{ 'interface' };
          ownedOperations += object uml20::kernel::features::Operation
                                                { name := 'Execute'; }
       }
}

query uml20::classes::Class::CreateM2BussinessLayerClasses() :
Set(uml20::classes::Class)
{
       Set
       {
           self.ToWAEClass(self.name + '_View', 'Server Page', ''),

           self.ToWAEClass(self.name + '_Action', 'Action', ''),

           self.ToDTOClass(self.name + '_InputTransfer', 'DTO Input'),

           self.ToDTOClass(self.name + '_OutputTransfer', 'DTO Output'),

           self.ToApplicationService()
       }
}

query uml20::classes::Class::CreateM2Dependencies(in NMMModel :
uml::together::Model) : Set(uml20::classes::Dependency)
{
       Set
       {
          NMMModel.CreateDependency(self, 'FacadeImp',
```

```
                                        'Application Service'),

            NMMModel.CreateDependency(self, 'Controller',
                                        'Server Page', 'Forward'),

            self.CreateDependency(NMMModel, 'Action', 'Facade'),

            self.CreateDependency(NMMModel, 'Action', 'action',
                                                    'Implement'),

            self.ToDTOOutputDependencies()
        }
}
```

5.3 Ejemplo

En las secciones anteriores se ha introducido y definido las reglas de transformación QVT para convertir modelos NMMp en modelos UML-WAE con arquitectura Model 2. En esta sección se utilizará nuevamente el ejemplo utilizado en el Capitulo 4 para producir de acuerdo a las reglas de transformación comentadas en la sección anterior el modelo UML-WAE con arquitectura Model 2 correspondiente. Es importante comentar que todas las observaciones hechas en la sección “Ejemplo” del Capitulo 4 son válidas también en esta sección, por lo tanto no se volverán a comentar aquí.

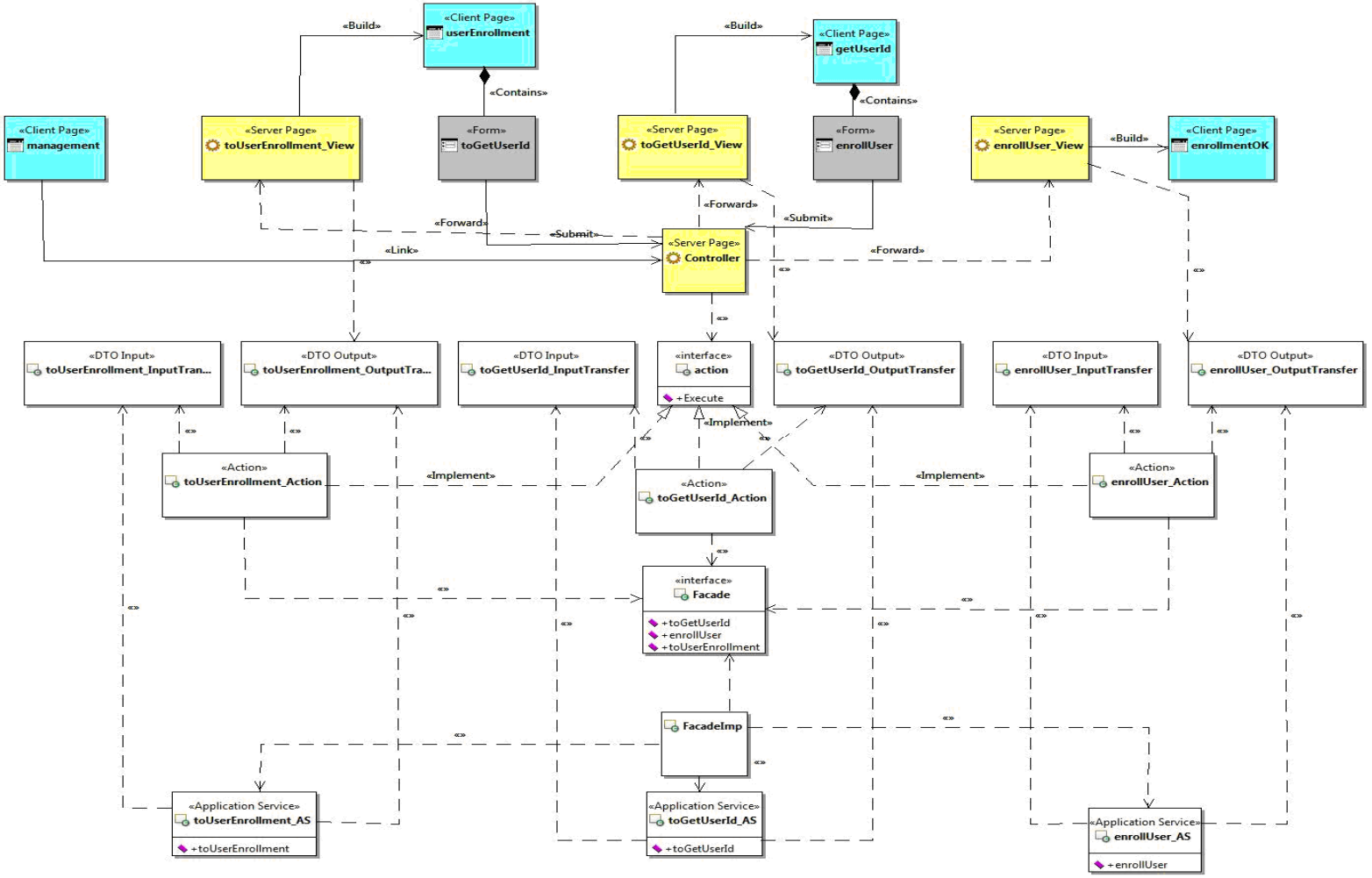
Al igual que en el ejemplo comentado en el Capitulo 4, la idea es que, partiendo del modelo NMMp de la figura 4-19 obtener su correspondiente modelo UML-WAE con arquitectura Model 2 mediante la aplicación automática de las reglas de transformación QVT.

En la figura 5-7 se muestra el modelo UML-WAE con arquitectura Model 2 obtenido. Como puede observarse las páginas NMMp han sido convertidas en páginas UML-WAE, las anclas computacionales han sido utilizadas para crear todos los componentes computacionales de la capa de negocio. Por ejemplo, la página `getUserId` de la figura 4-19 ha sido transformada en las páginas `getUserId_View`, `getUserId` y `enrollUser`, mientras el ancla computacional `enrollUser` ha sido transformado en las clases `enrollUser_InputTransfer`, `enrollUser_OutputTransfer`, `enrollUserAction`, `enrollUser_AS` y en el método `enrollUser` de la clase

`Facade`. Los enlaces entre anclas y páginas en el diagrama NMMp han sido transformados en enlaces entre los elementos UML-WAE.

Es importante señalar que mientras el diagrama NMMp de la figura 4-19 tiene solo siete clases UML (cuatro páginas y tres anclas computacionales), el diagrama UML-WAE de la figura 4-20 tiene veinticinco clases. Por lo tanto el poder semántico de los elementos NMMp queda demostrado una vez más.

Figura 5-7 Diagrama UML-WAE con arquitectura Model 2 generado automáticamente a partir del diagrama NMMp de la Figura 4-19 que modela la funcionalidad de inscripción de usuarios.



6. Conclusiones y trabajo futuro

El diseño de aplicaciones Web es una labor compleja. Una prueba de ello es el intenso trabajo realizado por la comunidad de la Ingeniería Web en los últimos años que ha producido un gran número de notaciones de diseño. Otra es el desarrollo de herramientas comerciales diseñadas especialmente para el diseño y construcción de aplicaciones Web.

En la actualidad, existen enfoques que cubren el desarrollo de cada uno de los aspectos de las aplicaciones Web usando notaciones abstractas de diseño. Aunque estas notaciones no están orientadas al desarrollo de código, éste es obtenido a partir de ellas usando herramientas específicas para tal fin. De esta forma, el mantenimiento y evolución de las aplicaciones Web queda ligado a la notación y herramienta específica utilizadas para su desarrollo. Por lo general, estos enfoques, que en este trabajo han sido llamados enfoques basados en notación, no utilizan patrones arquitectónicos ni de diseño.

Por otro lado, existen herramientas comerciales semejantes a entornos de desarrollo visual para lenguajes específicos, que, usando diagramas visuales, configuran el código de las aplicaciones Web. Aunque este enfoque está muy lejos de las notaciones de diseño abstracto, en la mayoría de los casos, el mantenimiento de la aplicación también está sujeto a la herramienta específica y la aplicación de patrones arquitectónicos depende de su inclusión explicita en la herramienta.

En un término medio existen otros dos enfoques. Uno se basa en el desarrollo de aplicaciones Web por medio de la utilización de UML estándar y patrones arquitectónicos y de diseño. En este trabajo dicho enfoque ha sido llamado enfoque basado en patrones. El inconveniente de este enfoque es que UML estándar carece de la semántica necesaria para expresar una descripción detallada de la capa de presentación de aplicaciones Web.

El otro enfoque intermedio usa notaciones abstractas para diseñar modelos que luego son convertidos a PSMs para facilitar la generación de código. Si los procesos de transformación de un modelo desarrollado con una notación abstracta a un modelo PSM, y del modelo PSM a código, es realizada utilizando transformaciones estándar, el mantenimiento y evolución de las aplicaciones no queda sujeto a ninguna herramienta. Además, si el enfoque permite incluir patrones arquitectónicos y de diseño, el

mantenimiento del software es mejorado. NMMp está en esta categoría. Aunque cada enfoque tiene ventajas y desventajas, NMMp pretende ser una notación equilibrada para la caracterización de mapas navegacionales de aplicaciones Web.

De acuerdo a nuestra experiencia, UML es adecuado para caracterizar el diseño de capas de negocio, de integración y de recursos en el desarrollo de aplicaciones Web, pero, es necesario un complemento que permita caracterizar la capa de presentación. UML-WAE intenta llenar este vacío, no obstante, sus diagramas de diseño, aunque muy valiosos desde el punto de vista de los desarrolladores, incluyen detalles arquitectónicos que los convierte en demasiado complejos para representar la estructura navegacional de las aplicaciones Web.

Los diagramas NMMp pueden ser usados como complemento a los diagramas UML-WAE para representar mapas navegacionales en *estado puro*, siendo por tanto adecuados para ser validados por los usuarios. Los modelos NMMp son diagramas abstractos, independientes de cualquier detalle arquitectónico y de cualquier lenguaje de programación. Son valiosos tanto para desarrolladores como para usuarios.

Los modelos NMMp, y los modelos UML-WAE a los cuales son transformados, caracterizan la capa de presentación de las aplicaciones Web, de forma que pueden ser un complemento directo para los diagramas UML estándar que caracterizan el resto de las capas de la aplicación. Los diagramas UML WAE generados incluyen patrones arquitectónicos, y los diagramas UML estándar pueden incluir el resto de patrones arquitectónicos. Esto permite un uso integrado de NMMp y UML, utilizando y promoviendo el uso de patrones arquitectónicos en cada capa. Además, como NMMp es una notación basada en UML, los diagramas de *todas* las capas de una aplicación Web pueden ser desarrollados utilizando una herramienta CASE de propósito general que soporte XMI y transformaciones QVT.

Finalmente, el código es obtenido usando reglas de transformación, explícitamente definidas por el usuario ó incluidas en la herramienta CASE de propósito general. Así, aunque el código generado no es tan completo como el generado por las herramientas del enfoque basado en notación, es mucho más mantenible y no esta ligado a ninguna notación ni herramienta específica.

Sin embargo, NMMp no contiene capacidades RIA y no proporciona aplicaciones Web *listas* para ser ejecutadas como la mayoría de los enfoques basados en notación. Las capacidades RIA podrían ser incluidas como en el resto de las notaciones

de diseño, pero debido a la naturaleza orientada al cliente intrínseca de las interfaces RIA, su inclusión en NMMp ha sido postergada. En cualquier caso, dado que la notación NMMp está centrada en los mapas navegacionales, una mezcla entre RUX y NMMp podría posibilitar una descripción detallada de las interfaces de usuario que incluyera *widgets* y mecanismos propios de las aplicaciones RIA, de forma semejante a como se detalla en (Preciado et al., 2008).

En cuanto a la generación de aplicaciones Web *listas* para ser ejecutadas, NMMp considera los diagramas de diseño como representaciones abstractas del código y no como simples representaciones visuales de código (Booch et al., 2007; Rumbaugh et al., 2010). De esta forma, con el enfoque NMMp, el código puede ser generado pero no la aplicación como tal.

El trabajo futuro incluye complementar las transformaciones QVT con restricciones OCL (OMG-OCL, 2011) que aseguren la validación sintáctica de los diagramas NMMp y la inclusión de características RIA en las primitivas de diseño NMMp. En este sentido, la posible integración de RUX con NMMp debería ser analizada. Finalmente, la aplicación del enfoque mixto comentado en este trabajo a otras capas de las aplicaciones Web está siendo llevado a cabo, considerando además, la configuración de los diseños con una importante variedad de patrones arquitectónicos.

Referencias

ABRAHAO, S., OLSINA, L. & PASTOR, O. 2003. Towards the quality evaluation of functional aspects of operative web applications. *Advanced Conceptual Modeling Techniques*, 2784**,** 325-338.

ALUR, D., CRUPI, J. & MALKS, D. 2003. *Core J2EE Patterns. Best Practices and Design Strategies*, Sun Microsystems Press, Prentice Hall.

APACHE. 2007. *Apache Struts* [Online]. Apache. Available: http://struts.apache.org/ [Accessed marzo 2011].

ARLOW, J. & NEUSTADT, I. 2005. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, Addison-Wesley Professional.

BERNSTEIN, M. 1998. Patterns of hypertext. *Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space---structure in hypermedia systems: links, objects, time and space---structure in hypermedia systems.* Pittsburgh, Pennsylvania, United States: ACM.

BEZIVIN, J. 2004. In search of a basic principle for model driven engineering. *UPGRADE: The European Journal for the Informatics Professional,* 5.

BOOCH, G., MAKSIMCHUK, R., ENGEL, M., YOUNG, B., CONALLEN, J. & HOUSTON, K. 2007. *Object-Oriented Analysis and Design with Applications*, Addison-Wesley.

BORLAND. 2011. *Borland Together CASE Tool* [Online]. Available: http://www.borland.com/us/products/together/index.aspx [Accessed marzo 2011].

BOZZON, A., COMAI, S., FRATERNALI, P. & CARUGHI, G. T. 2006. Conceptual modeling and code generation for rich internet applications. *Proceedings of the 6th international conference on Web engineering.* Palo Alto, California, USA: ACM.

BRAMBILLA, M., CERI, S., FRATERNALI, P. & MANOLESCU, I. 2006. Process modeling in Web applications. *ACM Trans. Softw. Eng. Methodol.*, 15**,** 360-409.

BROWN, S., BURDICK, R., FALKNER, J., GALBRAITH, B., JOHNSON, R., KIM, L., KOCHMER, C., KRISTMUNDSSON, T. & LI, S. 2002. *Professional JSP*, Birmingham.

BROWN, S., DALTON, S., JEPP, D., JOHNSON, D., LI, S. & RAIBLE, M. 2005. *Pro JSP 2*, Berkeley, CA.

BRUCKER, A. & DOSER, J. 2007. Metamodel-based UML Notations for Domain-specific Languages. *4th International Workshop on Language Engineering* [Online].

BUSCH, M. & KOCH, N. 2009. MagicUWE - A CASE Tool Plugin for Modeling Web Applications. *Web Engineering, Proceedings,* 5648**,** 505-508.

CERI, S., FRATERNALI, P. & BONGIO, A. 2000. Web Modeling Language (WebML): a modeling language for designing Web sites.

COMMUNITY, J. 2009. *JBoss* [Online]. RedHat. Available: http://www.jboss.org/ [Accessed marzo 2011].

CONALLEN, J. 1999. Modeling Web application architectures with UML. *Commun. ACM,* 42**,** 63-70.

CONALLEN, J. 2003. *Building Web applications with UML,* Boston, Addison-Wesley.

DISTANTE, D., PEDONE, P., ROSSI, G. & CANFORA, G. 2007. Model-driven development of Web applications with UWA, MVC and JavaServer Faces. *Web Engineering. Proceedings 7th International Conference, ICWE 2007***,** 457-72.

ECLIPSE. *Atlas Transformation Language* [Online]. Eclipse. Available: http://www.eclipse.org/atl/ [Accessed abril 2011].

ECLIPSE. *JET* [Online]. Eclipse. Available: http://www.eclipse.org/modeling/m2t/ [Accessed abril 2011].

ECLIPSE. 2009. *Eclipse Modeling Framework* [Online]. Eclipse. Available: http://www.eclipse.org/modeling/emf/ [Accessed marzo 2011].

ECLIPSE, F. 2011a. *Graphical Modeling Framework - GMF* [Online]. The Eclipse Foundation Community. Available: http://www.eclipse.org/modeling/gmp/ [Accessed Marzo 2011].

ECLIPSE, F. 2011b. *The Eclipse Foundation Open Source Community Website* [Online]. The Eclipse Foundation. Available: http://www.eclipse.org/ [Accessed Abril 2011].

ELSSAMADISY, A. 2006. Review of "Hibernate: A J2EE Developer's Guide by Will Iverson", Pearson Education Inc., 2005, ISBN: 0-471-20282-7. *SIGSOFT Softw. Eng. Notes,* 31**,** 42-43.

ERL, T. 2009b. *SOA Design Patterns,* Prentice Hall PTR.

FERNANDEZ, P. & NAVARRO, A. 2009. *Un Análisis Crítico a la Aproximación Model-Driven Architecture.* Proyecto Fin de Máster, Universidad Complutense de Madrid.

FOWLER, M. 2003. *Patterns of enterprise application architecture,* Boston, Addison-Wesley.

FRATERNALI, P. & PAOLINI, P. 1998. A Conceptual Model and a Tool Environment for Developing More Scalable, Dynamic, and Customizable Web Applications. *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology.* Springer-Verlag.

FREEMARKER. 2010. *FreeMarker Overview* [Online]. Freemarker. Available: http://freemarker.org/ [Accessed marzo 2011].

FUENTES, L. & VALLECILLO, A. 2004. Una Introducción a los Perfiles UML.

GAMMA, E., HELM, R., JOHNSON, R. & VLISSIDES, J. 2004. *Design patterns : elements of reusable object-oriented software*, Addison-Wesley.

GARDNER, T. & YUSUF, L. 2006. Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives.

GOMEZ, J., CACHERO, C. & PASTOR, O. 2001. Conceptual Modeling of Device-Independent Web Applications. *IEEE MultiMedia,* 8**,** 26-39.

GOOGLE. 2010. *Google Web Toolkit* [Online]. Google. Available: http://code.google.com/intl/es-ES/webtoolkit/ [Accessed Abril 2011].

GRANT, R. 2011. *Quick start guide to Oracle Fusion development: Oracle JDeveloper and Oracle ADF,* New York, McGraw-Hill.

HAILPERN, B. & TARR, P. 2006. Model-driven development: the good, the bad, and the ugly. *IBM Syst. J.,* 45**,** 451-461.

HARTWICH, C. & BERLIN, F. U. 2001. Why It Is So Difficult to Build N-Tiered Enterprise Applications.

HENNICKER, R. & KOCH, N. A UML-based methodology for hypermedia design. *In:* EVANS, A., KENT, S. & SELIC, B., eds. Proceedings of UML 2000. 3rd International Conference on the Unified Modeling Language, 2-6 October 2000 York, UK. Springer-Verlag, 410-424.

HENNICKER, R. & KOCH, N. 2001. Systematic design of Web applications with UML. *Unified modeling language.* IGI Publishing.

HERTEL, M. *Aspects of AJAX* [Online]. Available: http://www.mathertel.de/AJAX/AspectsOfAJAX0704.pdf [Accessed marzo 2011].

HIBERNATE. 2006. *Relational Persistence For Idiomatic Java* [Online]. RedHat. Available: http://www.hibernate.org/ [Accessed marzo 2011].

HUDSON, W. 2007. AJAX design and usability. *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it - Volume 2*. University of Lancaster, United Kingdom: British Computer Society.

IBM. 2009. *WebSphere Application Server (WAS)* [Online]. IBM. Available: http://www-01.ibm.com/software/webservers/appserv/was/ [Accessed marzo 2011].

IBM. 2010. *IBM Rational Software Architect Versión 8.0.2* [Online]. Available: http://publib.boulder.ibm.com/infocenter/rsahelp/v8/index.jsp?topic=/com.ibm.xtools.rsa_base.legal.doc/helpindex_rsa_base.html [Accessed marzo 2011].

ISAKOWITZ, T., STOHR, E. A. & BALASUBRAMANIAN, P. 1995. RMM: a methodology for structured hypermedia design. *Commun. ACM,* 38**,** 34-44.

KOCH, N. 2006. Transformation techniques in the model-driven development process of UWE. *Workshop proceedings of the sixth international conference on Web engineering*. Palo Alto, California: ACM.

KOCH, N. & KRAUS, A. 2002. The Expressive Power of UML-based Web Engineering.

KOCH, N. & KRAUS, A. 2003. Integration of Business Processes in Web Application Models.

KROISS, C., KOCH, N. & KNAPP, A. 2009. UWE4JSF: A Model-Driven Generation Approach for Web Applications. *In:* GAEDKE, M., GROSSNIKLAUS, M. & DIAZ, O. (eds.) *Web Engineering, Proceedings*. Berlin: Springer-Verlag Berlin.

KUMAR, B. V., NARAYAN, P. & NG, T. 2009. *Implementing SOA Using Java EE*, Addison-Wesley.

LINAJE, M., PRECIADO, J. C. & SANCHEZ-FIGUEROA, F. 2007. Engineering rich internet application user interfaces over legacy web models. *Ieee Internet Computing,* 11**,** 53-59.

LLORENTE, C. D. L. T., CASTRO, U. Z., NELSON, J. C. & BARROSO, M. A. R. 2011. *Guia de Arquitectura N-Capas Orientada al Dominio con .Net*, Microsoft Architecture.

MAGICUWE. 2010. *MagicUWE* [Online]. Available: http://uwe.pst.ifi.lmu.de/toolMagicUWE.html [Accessed abril 2011].

MANOLESCU, I., BRAMBILLA, M., CERI, S., COMAI, S. & FRATERNALI, P. 2005. Model-driven design and deployment of service-enabled Web applications. *ACM Transactions on Internet Technology,* 5**,** 439-79.

MELIA, S., GOMEZ, J., PEREZ, S. & DIAZ, O. 2010. Architectural and Technological Variability in Rich Internet Applications. *Ieee Internet Computing,* 14**,** 24-32.

MELIA, S., MARTINEZ, J.-J., PEREZ, A. & GOMEZ, J. 2009. OOH4RIA Tool: Una Herramienta basada en el Desarrollo Dirigido por Modelos para las RIAs.

MICROSYSTEMS, S. 2007. *Scaling The N-Tier Architecture* [Online]. Sun. Available: http://www.sun.com/software/whitepapers/wp-ntier/wp-ntier.pdf [Accessed marzo 2011].

MONDAY, P. B. 2003. *Web Services Patterns: Java Edition*, Apress.

MORENO, N., FRATERNALI, P. & VALLECILLO, A. 2007. WebML modelling in UML. *Iet Software,* 1**,** 67-80.

MURTHY, V. K. & SHI, N. 2003. Architectural Issues of Web-Based Electronic Business.

NAVARRO, A., FERNANDEZ-VALMAYOR, A., FERNANDEZ-MANJON, B. & SIERRA, J. L. 2008a. Characterizing navigation maps for web applications with the NMM approach. *Science of Computer Programming,* 71**,** 1-16.

NAVARRO, A., MERINO, J., FERNANDEZ-VALMAYOR, A. & CRISTOBAL, J. 2008c. Translating workflow diagrams into Web designs. *SEKE 2008. The 20th International Conference Proceedings on Software Engineering & Knowledge Engineering***,** 667-72.

NAVARRO, A., SIERRA, J. L., FERNANDEZ-VALMAYOR, A. & FERNANDEZ-MANJON, B. 2005. Conceptualization of Navigational Maps for Web Applications.

NAVARRO, A., CRISTOACUTEBAL, J., FERNAACUTENDEZ-VALMAYOR, A., FERNAACUTENDEZ, C., HERNANZ, H., GUILLOMIACUTEA, S. & BUENDIACUTEA, F. 2010. Towards a New Generation of Virtual Campuses. *Proceedings Sixth Advanced International Conference on Telecommunications (AICT 2010)*.

NETWORK, O. T. 2007. *Java EE Reference* [Online]. Oracle. Available: http://www.oracle.com/technetwork/java/javaee/documentation/index.html [Accessed marzo 2011].

NETWORK, O. T. 2009a. *JavaServer Faces Technology* [Online]. Oracle. Available: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html [Accessed marzo 2011].

NETWORK, S. D. 2009b. *Java 2 Platform Enterprise Edition - J2EE* [Online]. Available: http://java.sun.com/j2ee/overview.html [Accessed marzo 2011].

OMG. 2010. *Object Management Group - OMG* [Online]. OMG. Available: http://www.omg.org/ [Accessed marzo 2011].

OMG-MDA. 2003. *MDA Guide Version 1.0.1* [Online]. OMG. Available: http://www.omg.org/cgi-bin/doc?omg/03-06-01 [Accessed marzo 2011].

OMG-MOF. 2009. *Meta Object Facility (MOF)* [Online]. OMG. Available: http://www.omg.org/mof/ [Accessed marzo 2011].

OMG-OCL. 2011. *Object Constraint Language - OCL* [Online]. Object Management Group. Available: http://www.omg.org/spec/OCL/ [Accessed Marzo 2011].

OMG-QVT. 2009. *Query/View/Transformation (QVT)* [Online]. OMG. Available: http://www.omg.org/spec/QVT/1.0/ [Accessed marzo 2011].

OMG-UML. 2010a. *Unified Modeling Language (UML) Infrastructure* [Online]. Available: http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/ [Accessed marzo 2011].

OMG-UML. 2010b. *Unified Modeling Language (UML) Superstructure* [Online]. Available: http://www.omg.org/spec/UML/2.3/Superstructure/PDF/ [Accessed marzo 2011].

OMG-UML. 2010. *Unified Modeling Language - UML* [Online]. Object Management Group. Available: http://www.uml.org/ [Accessed Febrero 2011].

OMG-XMI. 2007. *XML Metadata Interchange (XMI)* [Online]. OMG. Available: http://www.omg.org/spec/XMI/2.1.1/ [Accessed marzo 2011].

OPENARCHITECTUREWARE. 2011. *Official OpenArchitectureWare HomePage* [Online]. OpenArchitectureWare. Available: http://www.openarchitectureware.org/ [Accessed Abril 2011].

OPENSYMPHONY. *Object-Graph Navigation Language* [Online]. Available: http://www.opensymphony.com/ognl/ [Accessed marzo 2011].

ORACLE. 2009. *Oracle Application Development Framework Overview* [Online]. Oracle White Pape. Available: http://www.oracle.com/technetwork/developer-tools/adf/adf-11-overview-1-129504.pdf [Accessed febrero 2011].

ORACLE. 2010a. *Oracle ADF Faces: Rich Client Components* [Online]. Oracle Data Sheet. Available: http://www.oracle.com/technetwork/developer-tools/adf/overview/index-092391.html [Accessed febrero 2011].

ORACLE. 2010b. *Oracle ADF: Key features and benefits* [Online]. Oracle Data Sheet. Available: http://www.oracle.com/technetwork/developer-tools/adf/adf11g-data-sheet-1-133847.pdf [Accessed febrero 2011].

ORACLE. 2010c. *Oracle Service-Oriented Architecture - SOA* [Online]. Oracle Data Sheet. Available: http://www.oracle.com/lad/products/middleware/soa/index.html [Accessed febrero 2011].

ORACLE. 2010d. *Oracle-Application Development Framework Oracle-ADF* [Online]. Oracle. Available: http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html [Accessed marzo 2011].

ORACLE. 2011. *Oracle Application Development Framework Overview* [Online]. Oracle White Pape. Available: http://www.oracle.com/technetwork/developer-tools/adf/adf-11-overview-1-129504.pdf [Accessed febrero 2011].

PASTOR, O., INSFRÁN, E., PELECHANO, V., ROMERO, J. & MERSEGUER, J. 1997. OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. *Proceedings of the 9th International Conference on Advanced Information Systems Engineering*. Springer-Verlag.

PEREZ, S., DIAZ, O., MELIA, S. & GOMEZ, J. 2008. Facing Interaction-Rich RIAs: The Orchestration Model. *Proceedings of the 2008 Eighth International Conference on Web Engineering*. IEEE Computer Society.

PIN-SHAN CHEN, P. 1976a. The entity-relationship model-toward a unified view of data. *ACM Transactions on Database Systems*, 1**,** 9-36.

PIN-SHAN CHEN, P. 1976b. The entity-relationship model-toward a unified view of data. *ACM Transactions on Database Systems|ACM Transactions on Database Systems*, 1**,** 9-36.

PRECIADO, J. C., LINAJE, M., MORALES-CHAPARRO, R., SANCHEZ-FIGUEROA, F., GEFEI, Z., KROISS, C. & KOCH, N. 2008. Designing rich Internet applications combining UWE and RUX-method. *2008 8th International Conference on Web Engineering (ICWE)***,** 148-54.

PRESSMAN, R. 2009. *Software Engineering: A Practitioners Approach*, McGraw-Hill.

RUMBAUGH, J., JACOBSON, I. & BOOCH, G. 2010. *Unified Modeling Language Reference Manual*, Addison-Wesley.

SHAN, T. C. & HUA, W. W. 2006. Solution Architecture for N-Tier Applications. *Proceedings of the IEEE International Conference on Services Computing*. IEEE Computer Society.

SOMMERVILLE, I. 2010. *Software Engineering*, Addison-Wesley.

SPAANJAARS, I. 2010. *Beginning ASP.NET 4: in C# and VB (Wrox Programmer to Programmer)*, Wiley Publishing, Inc.

STARKEY, M., ZHONG, G. Q., CHEN, L. L., YANG, C., ZHAN, E. & ZOU, L. 2008. Using IBM Rational Software Architect to develop Ajax-supported JavaServer Faces components. Available: http://www.ibm.com/developerworks/rational/library/08/0708_starkey/index.html.

SWITHIBANK, P., CHESSELL, M., GARDNER, T., GRIFFIN, C., MAN, J., WYLLE, H. & YUSUF, L. 2005. *Patterns: Model-Driven Development Using IBM Rational Software Architect*.

THOMPSON, L. & NOWICKI, S. D. 2009. *Professional PHP6 (Wrox Programmer to Programmer)*, Wrox.

W3C. *Extensible Stylesheet Language (XSL)* [Online]. Available: http://www.w3.org/TR/xsl/ [Accessed abril 2011].

W3C. *Web Services Description Language (WSDL)* [Online]. W3C. Available: http://www.w3.org/TR/wsdl [Accessed abril 2011].

W3C. 1999. *XSL Transformations (XSLT)* [Online]. Available: http://www.w3.org/TR/xslt [Accessed abril 2011].

W3C. 2005. *Document Object Model - DOM* [Online]. Available: http://www.w3.org/DOM/ [Accessed Marzo 2011].

W3C. 2008. *Cascading Style Sheets - CSS* [Online]. W3C. Available: http://www.w3c.es/divulgacion/guiasbreves/hojasestilo [Accessed julio 2011].

W3C. 2011a. *Extensible Markup Language - XML* [Online]. Available: http://www.w3.org/XML/ [Accessed Marzo 2011].

W3C. 2011b. *HyperText Markup Language - HTML* [Online]. Available: http://www.w3.org/html/ [Accessed Junio 2011].

WEAVER, J. L., MUKHAR, K., CRUME, J. P. & HORTON, I. 2004. *Beginning J2EE 1.4: From Novice to Professional*, Apress.

WEBRATIO. *WebRatio* [Online]. Available: http://www.webratio.com [Accessed abril 2011].

Apendice A – Transformaciones QVT de NMMp a UML-WAE Model 1

```
transformation T1Model_NMMp_To_Model_UML-WAE-Mode-1;


metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

mapping main(in NMMModel: uml::together::Model): uml::together::Model
{
    init
    {
      var NMMClasses := NMMModel.allInstances(uml20::classes::Class);

      var NMMDependencies :=
      NMMModel.allInstances(uml20::classes::Dependency);

      var WAEClasses := NMMClasses->collect(NMMClass |
      NMMClass.GetWAEClasses())->union(

      NMMModel.CreateM1Classes())->collect(c | SetTaggedValues(c));

      var WAEDependencies := NMMClasses->collect(NMMClass |
      NMMClass.GetWAEDependencies(NMMModel))->union(

      NMMDependencies->collect(NMMDependency |
      NMMDependency.GetWAEDependencies()))->union(
      NMMModel.GetModelDependencies())->collect(d |
      SetTaggedValues(d))
    }
    object
    {
      ownedMembers := WAEClasses->asOrderedSet();

      dependencies := WAEDependencies->asOrderedSet();

    }
}


query uml::together::Model::GetModelDependencies() :
Set(uml20::classes::Dependency)
{
      Set
      {
            self.CreateDependency(self, 'Controller', 'action', ''),

            self.CreateDependency(self, 'FacadeImp', 'Facade',
            'Implement')
      }
}
```

```
query uml::together::Model::CreateM1Classes() :
Set(uml20::classes::Class)
{
      if(self.allInstances(uml20::classes::Dependency)->select(cf |
      cf.stereotypes = OrderedSet{'Computing Function'})->notEmpty())
      then

            Set{ self.ToFacadeInterface(), self.ToFacadeClass() }

      else

            Set{}

      endif
}


mapping uml::together::Model::ToFacadeInterface() :
uml20::classes::Class
{
      init
      {
            var ComputingAnchorClass :=
            self.allInstances(uml20::classes::Class)->select(cf |
            cf.stereotypes = OrderedSet{'Form Computing Anchor'})->
            union(self.allInstances(uml20::classes::Class)->select(cf
            | cf.stereotypes = OrderedSet{'Non Form Computing
            Anchor'}))
      }
      object
      {
            name := 'Facade';
            stereotypes := OrderedSet{ 'interface' };
            ownedOperations := ComputingAnchorClass->collect(fca |
            AddOperationInClass(fca))->asOrderedSet();
      }
}


mapping uml::together::Model::ToFacadeClass() : uml20::classes::Class
{

      object
      {
            name := 'FacadeImp';

      }
}


query uml20::classes::Class::GetWAEClasses() :
Set(uml20::classes::Class)
{
      if (self.stereotypes = OrderedSet{'Lasting Page'} or
          self.stereotypes = OrderedSet{'Transient Page'}) then

          Set{ self.ToWAEClass(self.name, 'Client Page', '') }

      else

            if self.stereotypes = OrderedSet{'Form Computing Anchor'}
```

```
      then
                  self.CreateM1Classes()->
                  union(self.ToWAEClass(self.name, 'Form', '')-
                  >asSet())

            else

                  if self.stereotypes = OrderedSet{'Non Form Computing
                  Anchor'} then

                        self.CreateM1Classes()

                  else

                        if self.stereotypes = OrderedSet{'Window'}
                        then

                              Set { self.ToWAEClass(self.name,
                              'FrameSet', '') }

                        else

                              if self.stereotypes =
                              OrderedSet{'Region'} then

                                    Set { self.ToWAEClass(self.name,
                                    'Target',   self.getPropertyValue
                                    ('ProfileNMM_x__y_Region_
                                    x__y_DefaultPage')) }

                              else

                                    Set{}

                              endif

                        endif

                  endif

            endif

      endif
}


query uml20::classes::Class::CreateM1Classes() :
Set(uml20::classes::Class)
{
      Set
      {
            self.ToWAEClass(self.name + '_SP', 'Server Page', ''),

            self.ToDTOClass(self.name + '_InputTransfer', 'DTO
            Input'),

            self.ToDTOClass(self.name + '_OutputTransfer', 'DTO
            Output'),

            self.ToApplicationService()
```

```
	}
}
query uml20::classes::Class::GetWAEDependencies(in NMMModel :
uml::together::Model) : Set(uml20::classes::Dependency)
{
	if (self.stereotypes = OrderedSet{'Form Computing Anchor'}) then

		self.GetFormDependencies()-
		>union(self.ToSubmitDependencies()->asSet())-
		>union(self.CreateM1Dependencies(NMMModel))

	else

		if (self.stereotypes = OrderedSet{'Non Form Computing
		Anchor'}) then

			self.GetLinkDependencies()-
			>union(self.CreateM1Dependencies(NMMModel))

		else

			Set{}

		endif

	endif
}


query uml20::classes::Class::CreateM1Dependencies(in NMMModel :
uml::together::Model) : Set(uml20::classes::Dependency)
{
	Set
	{

		self.CreateDependency(self, 'Server Page', 'DTO Input',
		'', ''),

		self.CreateDependency(self, 'Server Page', 'DTO Output',
		'', ''),

		self.CreateDependency(self, 'Application Service', 'DTO
		Input', '', ''),

		self.CreateDependency(self, 'Application Service', 'DTO
		Output', '', ''),

		self.CreateDependency(NMMModel, 'Server Page', 'Facade'),

		NMMModel.CreateDependency(self, 'FacadeImp', 'Application
		Service')
	}

}


query uml20::classes::Dependency::GetWAEDependencies() :
Set(uml20::classes::Dependency)
{
	if (self.stereotypes = OrderedSet{'Retrieval Function'}) then
```

```
            self.GetLinkDependencies()

      else

            if (self.stereotypes = OrderedSet{'Computing Function'})
            then

                  Set{ self.ToBuildDependencies() }

            else

                  if (self.stereotypes = OrderedSet{'Contains'} and
                  self.client.oclAsType(uml20::classes::Class).
                  stereotypes = OrderedSet{'Region'} and
                  self.supplier.oclAsType(uml20::classes::Class).
                  stereotypes = OrderedSet{'Window'}) then

                        Set{ self.ToContainsDependencies() }

                  else

                        Set{}

                  endif

            endif

      endif
}


mapping SetTaggedValues(inout Dependency : uml20::classes::Dependency)
: uml20::classes::Dependency
{
      init
      {

            Dependency.setPropertyValue('ProfileWAE_x__y_
            Link_x__y_Target', Dependency.description);

            Dependency.setPropertyValue('ProfileWAE_x__y_
            Submit_x__y_Target', Dependency.description);

            Dependency.description := '';

            result := Dependency;
      }
}


mapping SetTaggedValues(inout Class : uml20::classes::Class) :
uml20::classes::Class
{
      init
      {
            Class.setPropertyValue('ProfileWAE_x__y_Target_
            x__y_DefaultPage', Class.description);

            Class.description := '';
```

```
            result := Class;
      }
}
mapping uml20::classes::Dependency::ToBuildDependencies() :
uml20::classes::Dependency
{
      init
      {
            result := self.client.oclAsType(uml20::classes::Class).
            CreateDependency(self.supplier.oclAsType(uml20::
            classes::Class), 'Server Page', 'Client Page', 'Build',
            '');
      }
}


query uml20::classes::Dependency::GetLinkDependencies() :
Set(uml20::classes::Dependency)
{
      self.client.allInstances(uml20::classes::Dependency)->select(dc
      | dc.stereotypes->includes('Contains'))->collect(d |
      d.supplier.oclAsType(uml20::classes::Class).CreateDependency
      (self.supplier.oclAsType(uml20::classes::Class), 'Client Page',
      'Client Page', 'Link', self.client.oclAsType(uml20::classes::
      Class).getPropertyValue('ProfileNMM_x__y_Retrieval
      _Anchor_x__y_Target')))->asSet();
}


mapping uml20::classes::Dependency::ToContainsDependencies() :
uml20::classes::Dependency
{
      init
      {
            result := self.client.oclAsType(uml20::classes::Class).
            CreateDependency(self.supplier.oclAsType(
            uml20::classes::Class), 'Target', 'FrameSet',
            'Contains', '');
      }
}


query uml20::classes::Class::GetFormDependencies() :
Set(uml20::classes::Dependency)
{
      self.dependencies->select(d | d.stereotypes-
      >includes('Contains'))->collect(dc |
      self.CreateDependency(dc.supplier.
      oclAsType(uml20::classes::Class),'Form',
      'Client Page','Contains', ''))->asSet();
}


mapping uml20::classes::Class::ToSubmitDependencies() :
uml20::classes::Dependency
{
      init
      {
            result := self.CreateDependency(self, 'Form','Server
            Page','Submit', self.getPropertyValue('ProfileNMM_x__y_
```

```
            Form_Computing_Anchor_x__y_Target'));
      }
}
query uml20::classes::Class::GetLinkDependencies() :
Set(uml20::classes::Dependency)
{

      self.dependencies->select(d | d.stereotypes-
      >includes('Contains'))->collect(dc |
      dc.supplier.oclAsType(uml20::classes::Class).
      CreateDependency(self, 'Client Page','Server Page','Link',
      self.getPropertyValue('ProfileNMM_x__y_Non_Form_
      Computing_Anchor_x__y_Target')))->asSet();
}


mapping uml20::classes::Class::CreateDependency(in Supplier :
uml20::classes::Class, in CStereotype : String, in SStereotype :
String, DStereotype : String, TaggedValue : String) :
uml20::classes::Dependency
{
      object
      {
            client      := self.resolve(uml20::classes::Class)-
            >select(cc | cc.stereotypes->includes(CStereotype))-
            >any(true);

            supplier    := Supplier.resolve(uml20::classes::Class)-
            >select(cs | cs.stereotypes->includes(SStereotype))-
            >any(true);

            stereotypes := OrderedSet{ DStereotype };

            description := TaggedValue;

      }
}


mapping uml20::classes::Class::CreateDependency(in Supplier :
uml::together::Model, in CStereotype : String, in SName : String) :
uml20::classes::Dependency
{
      object
      {
            client      := self.resolve(uml20::classes::Class)-
            >select(cc | cc.stereotypes->includes(CStereotype))-
            >any(true);

            supplier    := Supplier.resolve(uml20::classes::Class)-
            >select(cs | cs.name = SName)->any(true);

      }
}


mapping uml::together::Model::CreateDependency(in Supplier
:uml20::classes::Class, in CName : String, in SStereotype : String) :
uml20::classes::Dependency
{
```

```
	object
	{
		client		:= self.resolve(uml20::classes::Class)-
		>select(cc | cc.name =  CName)->any(true);
		supplier	:= Supplier.resolve(uml20::classes::Class)-
		>select(cs | cs.stereotypes->includes(SStereotype))-
		>any(true);
	}
}


mapping uml::together::Model::CreateDependency(in Supplier :
uml::together::Model, in CName : String, in SName : String,
DStereotype : String) : uml20::classes::Dependency
{
	object
	{
		client := self.resolve(uml20::classes::Class)->select(f |
		f.name = CName)->any(true);

		supplier := Supplier.resolve(uml20::classes::Class)-
		>select(f | f.name = SName)->any(true);

		stereotypes := OrderedSet{ DStereotype };

	}
}


mapping uml20::classes::Class::ToWAEClass(in Name : String, in
Stereotype : String, in Description : String) : uml20::classes::Class
{
	object
	{
		name := Name;

		stereotypes := OrderedSet { Stereotype };

		description := Description;
	}
}


mapping uml20::classes::Class::ToDTOClass(in Name : String, in
Stereotype : String) : uml20::classes::Class
when
{
	( self.stereotypes = OrderedSet{'Form Computing Anchor'} or
	self.stereotypes = OrderedSet{'Non Form Computing Anchor'} ) and
	self.dependencies->select(cf | cf.stereotypes =
	OrderedSet{'Computing Function'})->notEmpty()
}
{
	object
	{
		name := Name;

		stereotypes := OrderedSet{ Stereotype };
	}
}
```

```
mapping uml20::classes::Class::ToApplicationService() :
uml20::classes::Class
when
{
      ( self.stereotypes = OrderedSet{'Form Computing Anchor'} or
      self.stereotypes = OrderedSet{'Non Form Computing Anchor'} ) and
      self.dependencies->select(cf | cf.stereotypes =
      OrderedSet{'Computing Function'})->notEmpty()
}
{
      object
      {
            name := self.name + '_AS';
            stereotypes := OrderedSet{'Application Service'};
            ownedOperations := AddOperationInClass(self)-
            >asOrderedSet();
      }
}


mapping AddOperationInClass(in ComputingAnchorClass :
uml20::classes::Class) : uml20::kernel::features::Operation
when
{
      ComputingAnchorClass.dependencies->select(cf | cf.stereotypes =
      OrderedSet{'Computing Function'})->notEmpty()
}
{
      object
      {
            name := ComputingAnchorClass.name;
            visibility := uml::kernel::VisibilityKind::PUBLIC;
      }
}
```

Apéndice B - Transformaciones QVT de NMMp a UML-WAE Model 2

```
transformation T2Model_NMMp_To_Model_UML-WAE-Mode-2;


metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';


mapping main(in NMMModel: uml::together::Model): uml::together::Model
{
    init
    {
      var NMMClasses := NMMModel.allInstances(uml20::classes::Class);

      var NMMDependencies :=
      NMMModel.allInstances(uml20::classes::Dependency);

      var WAEClasses := NMMClasses->collect(NMMClass |
      NMMClass.GetWAEClasses())->union(
      NMMModel.GetModel2Classes())->collect(c | SetTaggedValues(c));

      var WAEDependencies := NMMClasses->collect(NMMClass |
      NMMClass.GetWAEDependencies(NMMModel))->union(

      NMMDependencies->collect(NMMDependency |
      NMMDependency.GetWAEDependencies(NMMModel)))->union(

      NMMModel.GetModelDependencies())->collect(d |
      SetTaggedValues(d));
    }
    object
    {
      ownedMembers := WAEClasses->asOrderedSet();

      dependencies := WAEDependencies->asOrderedSet();

    }
}


query uml::together::Model::GetModelDependencies() :
Set(uml20::classes::Dependency)
{
      Set
      {
            self.CreateDependency(self, 'Controller', 'action', ''),

            self.CreateDependency(self, 'FacadeImp', 'Facade',
            'Implement')
      }
}
```

```
query uml::together::Model::GetModel2Classes() :
Set(uml20::classes::Class)
{
	if(self.allInstances(uml20::classes::Dependency)->select(cf |
	cf.stereotypes = OrderedSet{'Computing Function'})->notEmpty())
	then

		Set{ self.ToFacadeInterface(), self.ToFacadeClass(),
		self.ToActionClass(), self.ToControllerClass()}

	else

		if(self.allInstances(uml20::classes::Dependency)-
		>select(cf | cf.stereotypes = OrderedSet{'Retrieval
		Function'})->notEmpty()) then

			Set{ self.ToControllerClass() }

		else

			Set{}

		endif

	endif
}


mapping uml::together::Model::ToFacadeInterface() :
uml20::classes::Class
{
	init
	{
		var ComputingAnchorClass :=
		self.allInstances(uml20::classes::Class)->select(cf |
		cf.stereotypes = OrderedSet{'Form Computing Anchor'})-
		>union(

		self.allInstances(uml20::classes::Class)->
		select(cf | cf.stereotypes = OrderedSet{'Non Form
		Computing Anchor'}))
	}
	object
	{
		name := 'Facade';
		stereotypes := OrderedSet{ 'interface' };
		ownedOperations := ComputingAnchorClass->collect(fca |
		AddOperationInClass(fca))->asOrderedSet();
	}
}


mapping uml::together::Model::ToFacadeClass() : uml20::classes::Class
{

	object
	{
		name := 'FacadeImp';

	}
}
```

```
mapping uml::together::Model::ToActionClass() : uml20::classes::Class
{
      object
      {
            name := 'action';
            stereotypes := OrderedSet{ 'interface' };
            ownedOperations += object
            uml20::kernel::features::Operation{ name := 'Execute'; }
      }
}


mapping uml::together::Model::ToControllerClass() :
uml20::classes::Class
{
      object
      {
            name := 'Controller';
            stereotypes := OrderedSet{ 'Server Page' };
      }
}


query uml20::classes::Class::GetWAEClasses() :
Set(uml20::classes::Class)
{
      if (self.stereotypes = OrderedSet{'Lasting Page'} or
          self.stereotypes = OrderedSet{'Transient Page'}) then

          Set{ self.ToWAEClass(self.name, 'Client Page', '') }

      else

            if self.stereotypes = OrderedSet{'Form Computing Anchor'}
            then

                  self.CreateM2Classes()-
                  >union(self.ToWAEClass(self.name, 'Form', '')-
                  >asSet())

            else

                  if self.stereotypes = OrderedSet{'Non Form Computing
                  Anchor'} then

                        self.CreateM2Classes()

                  else
                        if self.stereotypes = OrderedSet{'Window'}
                        then

                              Set { self.ToWAEClass(self.name,
                              'FrameSet', '') }

                        else

                        if self.stereotypes =  OrderedSet{'Region'}
                        then

                                    Set { self.ToWAEClass(self.name,
                                     'Target',
```

```
                                    self.getPropertyValue(
                                    'ProfileNMM_x__
                                    y_Region_x__y_DefaultPage')) }

                              else

                                    Set{}

                              endif

                        endif

                  endif

            endif

      endif
}


query uml20::classes::Class::CreateM2Classes() :
Set(uml20::classes::Class)
{
      Set
      {
            self.ToWAEClass(self.name + '_View', 'Server Page', ''),

            self.ToWAEClass(self.name + '_Action', 'Action', ''),

            self.ToDTOClass(self.name + '_InputTransfer', 'DTO
            Input'),

            self.ToDTOClass(self.name + '_OutputTransfer', 'DTO
            Output'),

            self.ToApplicationService()
      }
}


query uml20::classes::Class::GetWAEDependencies(in NMMModel :
uml::together::Model) : Set(uml20::classes::Dependency)
{
      if (self.stereotypes = OrderedSet{'Form Computing Anchor'}) then


          Set { self.ToSubmitDependencies(NMMModel) }-
          >union(self.GetFormDependencies())
          ->union( self.CreateM2Dependencies(NMMModel) )

      else

            if (self.stereotypes = OrderedSet{'Non Form Computing
            Anchor'}) then

                  self.GetLinkDependencies(NMMModel,
                  self.getPropertyValue('ProfileNMM_x_
                  _y_Non_Form_Computing_Anchor_x__y_Target'))
                  ->union( self.CreateM2Dependencies(NMMModel) )

            else
```

```
                  if (self.stereotypes = OrderedSet{'Retrieval
                  Anchor'}) then

                        self.GetLinkDependencies(NMMModel,
                        self.getPropertyValue('ProfileNMM_x_
                        _y_Retrieval_Anchor_x__y_Target'))

                  else

                        Set{}

                  endif

            endif

      endif
}


query uml20::classes::Class::CreateM2Dependencies(in NMMModel :
uml::together::Model) : Set(uml20::classes::Dependency)
{
      Set
      {

            NMMModel.CreateDependency(self, 'FacadeImp', 'Application
            Service', ''),

            NMMModel.CreateDependency(self, 'Controller', 'Server
            Page', 'Forward'),

            self.CreateDependency(NMMModel, 'Action', 'Facade', '',
            ''),

            self.CreateDependency(NMMModel, 'Action', 'action',
            'Implement', ''),

            self.ToDTOOutputDependencies()
      }

      ->union(self.GetActionDependencies())-
      >union(self.GetApplicationServiceDependencies())
}


query uml20::classes::Dependency::GetWAEDependencies(in NMMModel :
uml::together::Model) : Set(uml20::classes::Dependency)
{
      if (self.stereotypes = OrderedSet{'Retrieval Function'}) then

            Set{ self.ToForwardDependencies(NMMModel) }

      else

            if (self.stereotypes = OrderedSet{'Computing Function'})
            then

                  Set{ self.ToBuildDependencies() }

            else
```

```
                  if (self.stereotypes = OrderedSet{'Contains'} and
                  self.client.oclAsType(uml20::classes::Class).
                  stereotypes = OrderedSet{'Region'} and
                  self.supplier.oclAsType(uml20::classes::Class).
                  stereotypes = OrderedSet{'Window'}) then

                        Set{ self.ToContainsDependencies() }

                  else

                        Set{}

                  endif

            endif

      endif
}


mapping SetTaggedValues(inout Dependency : uml20::classes::Dependency)
: uml20::classes::Dependency
{
      init
      {
            Dependency.setPropertyValue('ProfileWAE_x__y_Link_x_
            _y_Target', Dependency.description);

            Dependency.setPropertyValue('ProfileWAE_x_
            _y_Submit_x__y_Target', Dependency.description);

            Dependency.description := '';

            result := Dependency;
      }
}


mapping SetTaggedValues(inout Class : uml20::classes::Class) :
uml20::classes::Class
{
      init
      {

            Class.setPropertyValue('ProfileWAE_x__y_Target_x_
            _y_DefaultPage', Class.description);

            Class.description := '';

            result := Class;
      }
}


mapping uml20::classes::Dependency::ToBuildDependencies() :
uml20::classes::Dependency
{
      init
      {
            var Client :=
```

```
            self.client.oclAsType(uml20::classes::Class);

            result := Client.CreateDependency(self.supplier.
            oclAsType(uml20::classes::Class), 'Server Page', 'Client
            Page', 'Build', '');
      }
}


mapping uml20::classes::Dependency::ToForwardDependencies(in NMMModel
: uml::together::Model) : uml20::classes::Dependency
{

      init
      {
            var Supplier :=
            self.supplier.oclAsType(uml20::classes::Class);

            result := NMMModel.CreateDependency(Supplier,
            'Controller', 'Client Page', 'Forward');
      }
}


mapping uml20::classes::Dependency::ToContainsDependencies() :
uml20::classes::Dependency
{
      init
      {
            var Client :=
            self.client.oclAsType(uml20::classes::Class);

            result := Client.CreateDependency(self.supplier.
            oclAsType(uml20::classes::Class),
            'Target', 'FrameSet', 'Contains', '');
      }
}


query uml20::classes::Class::GetApplicationServiceDependencies() :
Set(uml20::classes::Dependency)
{
      Set { self.CreateDependency(self, 'Application Service','DTO
      Input','', ''), self.CreateDependency(self, 'Application
      Service','DTO Output','', '') };
}


query uml20::classes::Class::GetActionDependencies() :
Set(uml20::classes::Dependency)
{
      Set { self.CreateDependency(self, 'Action','DTO Input','', ''),
      self.CreateDependency(self, 'Action','DTO Output','', '') };

}


mapping uml20::classes::Class::ToDTOOutputDependencies() :
uml20::classes::Dependency
{
      init
```

```
	{
		result := self.CreateDependency(self, 'Server Page','DTO
		Output','', '');
	}
}


query uml20::classes::Class::GetFormDependencies() :
Set(uml20::classes::Dependency)
{
	self.dependencies->select(d | d.stereotypes-
	>includes('Contains'))->collect(dc |
	self.CreateDependency(dc.supplier.oclAsType(uml20::
	classes::Class),'Form','Client Page','Contains', ''))
	->asSet();
}


mapping uml20::classes::Class::ToSubmitDependencies(in NMMModel :
uml::together::Model) : uml20::classes::Dependency
{
	init
	{
		result := self.CreateDependency(NMMModel,
		'Form','Controller','Submit',
		self.getPropertyValue('ProfileNMM_x__y_Form_
		Computing_Anchor_x__y_Target'));
	}
}


query uml20::classes::Class::GetLinkDependencies(in NMMModel :
uml::together::Model, in TaggedValue : String) :
Set(uml20::classes::Dependency)
{

	self.dependencies->select(d | d.stereotypes-
	>includes('Contains'))->collect(dc |
	dc.supplier.oclAsType(uml20::classes::Class).
	CreateDependency(NMMModel,'Client Page','Controller','Link',
	TaggedValue))->asSet();
}


mapping uml20::classes::Class::CreateDependency(in Supplier :
uml20::classes::Class, in CStereotype : String, in SStereotype :
String, DStereotype : String, TaggedValue : String) :
uml20::classes::Dependency
{
	object
	{
		client      := self.resolve(uml20::classes::Class)-
		>select(cc | cc.stereotypes->includes(CStereotype))-
		>any(true);

		supplier    := Supplier.resolve(uml20::classes::Class)-
		>select(cs | cs.stereotypes->includes(SStereotype))-
		>any(true);

		stereotypes := OrderedSet{ DStereotype };
```

```
            description := TaggedValue;

      }
}


mapping uml20::classes::Class::CreateDependency(in Supplier :
uml::together::Model, in CStereotype : String, in SName : String,
DStereotype : String, Description : String) :
uml20::classes::Dependency
{
      object
      {
            client      := self.resolve(uml20::classes::Class)-
            >select(cc | cc.stereotypes->includes(CStereotype))-
            >any(true);

            supplier    := Supplier.resolve(uml20::classes::Class)-
            >select(cs | cs.name = SName)->any(true);

            stereotypes := OrderedSet{ DStereotype };

            description := Description;

      }
}


mapping  uml::together::Model::CreateDependency(in Supplier :
uml20::classes::Class, in CName : String, in SStereotype : String,
DStereotype : String) : uml20::classes::Dependency
{
      object
      {
            client      := self.resolve(uml20::classes::Class)-
            >select(cc | cc.name = CName)->any(true);

            supplier    := Supplier.resolve(uml20::classes::Class)-
            >select(cs | cs.stereotypes->includes(SStereotype))-
            >any(true);

            stereotypes := OrderedSet{ DStereotype };

      }
}


mapping uml::together::Model::CreateDependency(in Supplier :
uml::together::Model, in CName : String, in SName : String,
DStereotype : String) : uml20::classes::Dependency
{
      object
      {
            client := self.resolve(uml20::classes::Class)->select(f |
            f.name = CName)->any(true);

            supplier := Supplier.resolve(uml20::classes::Class)-
            >select(f | f.name = SName)->any(true);

            stereotypes := OrderedSet{ DStereotype };
```

```
	}
}


mapping uml20::classes::Class::ToWAEClass(in Name : String, in
Stereotype : String, in Description : String) : uml20::classes::Class
{
	object
	{
		name := Name;

		stereotypes := OrderedSet { Stereotype };

		description := Description;
	}
}


mapping uml20::classes::Class::ToDTOClass(in Name : String, in
Stereotype : String) : uml20::classes::Class
when
{
	( self.stereotypes = OrderedSet{'Form Computing Anchor'} or
	self.stereotypes = OrderedSet{'Non Form Computing Anchor'} ) and
	self.dependencies->select(cf | cf.stereotypes =
	OrderedSet{'Computing Function'})->notEmpty()
}
{
	object
	{
		name := Name;

		stereotypes := OrderedSet{ Stereotype };
	}
}


mapping uml20::classes::Class::ToApplicationService() :
uml20::classes::Class
when
{
	( self.stereotypes = OrderedSet{'Form Computing Anchor'} or
	self.stereotypes = OrderedSet{'Non Form Computing Anchor'} ) and
	self.dependencies->select(cf | cf.stereotypes =
	OrderedSet{'Computing Function'})->notEmpty()
}
{
	object
	{
		name := self.name + '_AS';

		stereotypes := OrderedSet{'Application Service'};

		ownedOperations := AddOperationInClass(self)-
		>asOrderedSet();
	}
}


mapping AddOperationInClass(in ComputingAnchorClass :
uml20::classes::Class) : uml20::kernel::features::Operation
```

```
when
{
	ComputingAnchorClass.dependencies->select(cf | cf.stereotypes =
	OrderedSet{'Computing Function'})->notEmpty()
}
{
	object
	{
		name := ComputingAnchorClass.name;
		visibility := uml::kernel::VisibilityKind::PUBLIC;
	}
}
```